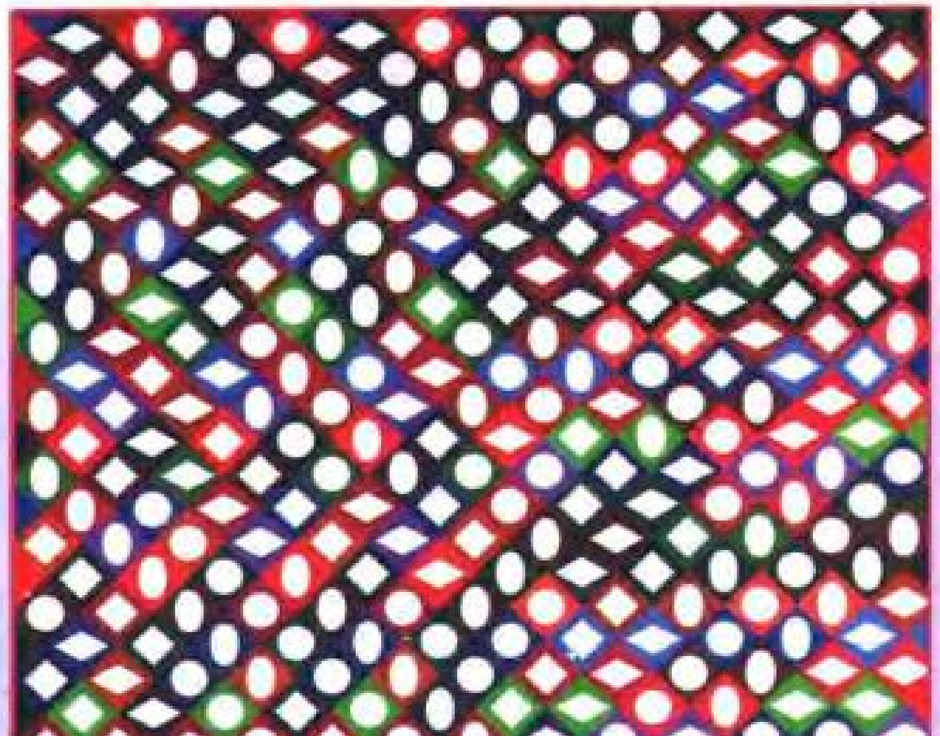
国外著名高等院校
信息科学与技术优秀教材

密码学概论

Introduction to CRYPTOGRAPHY with CODING THEORY



〔美〕Wade Trappe
Lawrence C.Washington 著

邹红霞 许鹏文 李勇奇 译

中文版

人民邮电出版社
POSTS & TELECOM PRESS

封面设计：胡平利

国外著名高等院校信息科学与技术优秀教材

PEARSON Prentice Hall

密码学概论

Introduction to CRYPTOGRAPHY *with* CODING THEORY

www.PearsonEd.com

ISBN 7-115-12184-2

9 787115 121844 >

人民邮电出版社网址 www.ptpress.com.cn

ISBN7-115-12184-2/TP·3914
定价：38.00元

国外著名高等院校信息科学与技术优秀教材

密码学概论

[美] Wade Trappe
Lawrence C.Washington 著
邹红霞 许鹏文 李勇奇 译

人 民 邮 电 出 版 社

图书在版编目（CIP）数据

密码学概论/（美）特拉普（Trappe，W.），（美）华盛顿（Washington，L.C.）著；许鹏文，邹红霞，李勇奇译. —北京：人民邮电出版社，2004.6
国外著名高等院校信息科学与技术优秀教材
ISBN 7－115－12184－2

Ⅰ. 密… Ⅱ. ①特… ②华… ③许… ④邹… ⑤李… Ⅲ. 密码—理论—高等学校—教材
Ⅳ. TN918.1

中国版本图书馆 CIP 数据核字（2004）第 021915 号

版 权 声 明

Simplified Chinese edition Copyright © 2003 by PEARSON EDUCATION ASIA LIMITED
and POSTS & TELECOMMUNICATIONS PRESS.
Introduction to Cryptography with Coding Theory(0130618144)
By Wade Trappe, Lawrence C.Washington
Copyright © 2002
All Rights Reserved.
Published by arrangement with the original publisher,Pearson Education,Inc.,publishing as Addison Wesley.
This edition is authorized for sale only in People's Republic of China (excluding the Special Administrative Region of Hong Kong and Macao).
本书封面贴有 Pearson Education（培生教育出版集团）激光防伪标签，无标签者不得销售。

国外著名高等院校信息科学与技术优秀教材

密码学概论

◆ 著 [美] Wade Trappe Lawrence C.Washington
译 邹红霞 许鹏文 李勇奇
责任编辑 李 际

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街 14 号
邮编 100061 电子函件 315@ptpress.com.cn
网址 http://www. ptpress.com.cn
读者热线 010-67132705
北京汉魂图文设计有限公司制作
北京隆昌伟业印刷有限公司印刷
新华书店总店北京发行所经销

◆ 开本：787×1092 1/16
印张：21.75
字数：526 千字 2004 年 6 月第 1 版
印数：1-4 000 册 2004 年 6 月北京第 1 次印刷

著作权合同登记 图字：01－2002－5927 号

ISBN 7-115-12184-2/TP · 3914

定价：38.00 元

本书如有印装质量问题，请与本社联系 电话：（010）67129223

内容提要

本书全面讲解了密码学的基本知识以及相关的基础数论，并对椭圆曲线、量子密码体制等密码学前沿知识进行了介绍。在此基础上，本书对数字签名、数字现金等应用问题作了较为详细的阐述。另外，本书每章都给出了相应的习题，而且在附录中给出了用 Mathematica、Maple 和 MATLAB 实现的相关示例。

本书可供高等院校应用数学、通信和计算机等专业用作密码学、通信安全和网络安全等课程的教材或参考书，也可供信息安全系统设计开发人员、密码学和信息安全爱好者参考。

出版说明

2001年，教育部印发了《关于“十五”期间普通高等教育教材建设与改革的意见》。该文件明确指出，“九五”期间原国家教委在“抓好重点教材，全面提高质量”方针指导下，调动了各方面的积极性，产生了一大批具有改革特色的新教材。然而随着科学技术的飞速发展，目前高校教材建设工作仍滞后于教学改革的实践，一些教材内容陈旧，不能满足按新的专业目录修订的教学计划和课程设置的需要。为此该文件明确强调，要加强国外教材的引进工作。当前，引进的重点是信息科学与技术和生物科学与技术两大学科的教材。要根据专业（课程）建设的需要，通过深入调查、专家论证，引进国外优秀教材。要注意引进教材的系统配套，加强对引进教材的宣传，促进引进教材的使用和推广。

邓小平同志早在1977年就明确指出：“要引进外国教材，吸收外国教材中有益的东西。”随着我国加入WTO，信息产业的国际竞争将日趋激烈，我们必须尽快培养出大批具有国际竞争能力的高水平信息技术人才。教材是一个很关键的问题，国外的一些优秀教材不但内容新，而且还提供了很多新的研究方法和思考方式。引进国外原版教材，可以促进我国教学水平的提高，提高学生的英语水平和学习能力，保证我们培养出的学生具有国际水准。

为了贯彻中央“科教兴国”的方针，配合国内高等教育教材建设的需要，人民邮电出版社约请有关专家反复论证，与国外知名的教材出版公司合作，陆续引进一些信息科学与技术优秀教材。第一批教材针对计算机专业的主干核心课程，是国外著名高等院校所采用的教材，教材的作者都是在相关领域享有盛名的专家教授。这些教材内容新，反映了计算机科学技术的最新发展，对全面提高我国信息科学与技术的教学水平必将起到巨大的推动作用。

出版国外著名高等院校信息科学与技术优秀教材的工作将是一个长期的、坚持不懈的过程，我社网站（www.ptpress.com.cn）上介绍了我们陆续推出的图书的详细情况，敬请关注。希望广大教师和学生将使用中的意见和建议及时反馈给我们，我们将根据您的反馈不断改进我们的工作，推出更多更好的引进版信息科学与技术教材。

人民邮电出版社

前　言

自 1997 年以来，本书一直作为马里兰大学高年级（大三和大四年级）本科生密码学课程的教材。该教材具有如下特点：

- 书中涉及到了最新的技术和广泛的数学基础。
- 未学过数论及计算机程序设计的学生可直接阅读本教材。
- 书中列举了大量的例子来验证算法的实际工作。

本书避免孤立讲述 RSA 算法及大量涉及到数论知识的离散对数等内容，也未提及具体的协议及怎样攻击别人的计算机。本书以描述性为主，涉及少量的数学证明。

该教材全面讲述了密码学的大量基础知识。本书中的许多章节内容超出了一学期的教学内容。前 8 章是课程的基础和核心部分，剩余章节可根据学生的层次选择讲解。

由于教材内容较多，按章节排出了序号，但除了第 3 章讲述的是该课程的基础数论知识以外，其余各章基本上都是独立的，可根据需要自由组织顺序，虽然我们不主张这样。有基础的读者可跳过前 3 章，直接阅读第 4～17 章。

信息论、椭圆曲线、量子密码体制及纠错码这几章比之前几章更趋于数学讨论。纠错码这章列在本教材中是有原因的，因为该章包括了密码术和广泛使用的编码理论。

计算机实例。考虑给出一个 RSA 实例，你可以选择两个 1 位素数，而伪装成用 50 位的素数，或者选择你熟悉的软件包用大的素数来实现实例。也可以考虑采用移位密码，尝试所有 26 个英文字母的各种移位情况，去解密一段消息，显然这需要借助于计算机。本书的最后附上了用 Mathematica、Maple 和 MATLAB 语言写出的程序实例，之所以选择这 3 种语言，是因为它们比较容易且不要求编程人员有很多的编程经验。即使没有计算机上机操作也可以学习该课程，但这些实例作为书中完整的一部分被列出，应该尽可能地学习它们。这些实例不仅包括怎样去实现某个数学示例和计算，而且证明了书中所提出的重要观点和问题。为了保持本书的逻辑性和连续性，我们还在每章的最后给出了用这 3 种语言所

写的计算机实例。

程序源代码可到如下网站下载：

www.prenhall.com/washington

上课讲解时，需将源程序安装至计算机（其中至少要安装一种语言），为保证效果，需要利用投影仪投影程序执行的结果。课后作业（每章后的上机习题）可基于一种软件让学生自己练习。当然，学生也可以选择他们熟悉的程序语言来替代。

致谢。在本书的编写过程中许多人提供了大量的帮助。首先，要感谢我的学生，他们无私地、热情地为本书提出了许多宝贵的意见。我要特别感谢 David Bindel，Jason Ernst，Christine Planchak，Haw-ren Fang，Marwan Oweis，Bob Grafton，他们收集了大量的资料并进行了录入。我的同事 Bill Gasarch 帮助校正了修订版，他的许多建议使我受益匪浅。Jonathan Rosenberg 和 Tim Strobell 提出了相当有价值的技术帮助。另外还要特别感谢 David Grant（Boulder 的 Colorado 大学），David M.Pozar（Amherst，Massachusetts 大学），Jugal K.Kalita（Colorado Springs 的 Colorado 大学）……，他们始终如一地在内容的组织与安排上提出了宝贵的建议。我们也很高兴与 Prentice Hall 的同仁们，特别是数学专家 George Lobell 和应用专家 Jeanne Audinor 的合作。

在此第一作者还要感谢 Nisha Gilra 提供的许多鼓励和宝贵意见，以及 Sheilagh O'Hare 和 K.J.Ray Liu 的支持。

第二作者要感谢 Susan Zengerle 和 Patrick Washington 在成书过程中的耐心、帮助和鼓励。

Wade Trappe

wxt@math.umd.edu

Lawrence C.Washington

lcw@math.umd.edu

目 录

第 1 章 密码学及其应用概述 …… 1

1.1 安全通信 …… 2

1.1.1 可能的攻击 …… 2

1.1.2 对称和公开密钥算法 …… 3

1.1.3 密钥长度 …… 5

1.2 密码学应用 …… 6

第 2 章 古典密码体制 …… 8

2.1 移位密码 …… 8

2.2 仿射密码 …… 9

2.3 Vigenère 密码 …… 11

2.3.1 发现密钥长度 …… 12

2.3.2 发现密钥：第一种方法 …… 13

2.3.3 发现密钥：第二种方法 …… 15

2.4 替换密码 …… 16

2.5 福尔摩斯密码 …… 18

2.6 Playfair 和 ADFGX 密码 …… 21

2.7 分组密码 …… 23

2.8 二进制数和 ASCII …… 26

2.9 一次一密 …… 27

2.10 伪随机序列生成 …… 28

2.11 线性反馈移位寄存序列 …… 30

2.12 Enigma …… 34

2.13 习题 …… 37

2.14 上机题 …… 39

第 3 章 基础数论 …… 42

3.1 基本概念 …… 42

3.1.1 整除 …… 42

3.1.2 素数 …… 43

3.1.3 最大公约数(Greatest Common Divisor) …… 44

3.2 求解 $ax+by=d$ …… 46

3.3 同余 …… 47
3.3.1 除法 …… 49
3.3.2 求 a-1(mod n) …… 50
3.3.3 当 gcd(a, n)=1 时，解 $ax\equiv c$(mod n) …… 50
3.3.4 如果 gcd(a, n)>1 怎么办 …… 50
3.3.5 分数的计算 …… 51
3.4 中国剩余定理 …… 51
3.5 模的幂计算 …… 53
3.6 费尔马小定理和欧拉定理 …… 54
3.7 本原根 …… 56
3.8 模 n 逆矩阵 …… 57
3.9 模 n 平方根 …… 58
3.10 有限域 …… 59
3.10.1 除法 …… 62
3.10.2 LFSR 序列 …… 64
3.11 习题 …… 65
3.12 上机题 …… 67

第 4 章 数据加密标准 …… 69

4.1 概述 …… 69
4.2 一个简单的类 DES 算法 …… 70
4.3 微分密码分析法 …… 72
4.3.1 具有三轮循环的微分密码分析法 …… 73
4.3.2 具有四轮循环的微分密码分析法 …… 75
4.4 DES …… 76
4.5 操作模式 …… 82
4.5.1 电子密码本（ECB） …… 82
4.5.2 密码分组链（CBC） …… 82
4.5.3 密码反馈（CFB） …… 83
4.6 破解 DES …… 84
4.7 口令的安全 …… 87
4.8 习题 …… 88

第 5 章 高级加密标准：Rijndael …… 90

5.1 基本算法 …… 90
5.2 层 …… 91
5.2.1 字节转换 …… 91
5.2.2 移动行变换 …… 92
5.2.3 混合列变换 …… 92

5.2.4 加循环密钥 …… 93
5.2.5 密钥计划表 …… 93
5.2.6 S-盒的构成 …… 94
5.3 解密 …… 94
5.4 设计中要考虑的问题 …… 96

第 6 章 RSA 算法 …… 98

6.1 RSA 算法 …… 98
6.2 对 RSA 的攻击 …… 101
6.3 素数判定 …… 103
6.4 因数分解 …… 106
6.5 RSA 挑战 …… 110
6.6 协议验证上的应用 …… 111
6.7 公钥概念 …… 111
6.8 习题 …… 113
6.9 上机题 …… 115

第 7 章 离散对数 …… 117

7.1 离散对数 …… 117
7.2 离散对数的计算 …… 118
7.2.1 Pohlig-Hellman 算法 …… 118
7.2.2 指数微积分 …… 120
7.2.3 模 4 离散对数的计算 …… 121
7.3 比特约定 …… 122
7.4 ElGamal 公钥体制 …… 123
7.5 习题 …… 124
7.6 上机题 …… 125

第 8 章 数字签名 …… 126

8.1 RSA 签名 …… 126
8.2 ElGamal 签名方案 …… 127
8.3 散列函数 …… 129
8.4 生日攻击 …… 132
8.4.1 签名方案中的生日攻击 …… 133
8.4.2 基于离散对数的生日攻击 …… 133
8.4.3 双重加密的中间相遇攻击 …… 134
8.5 数字签名算法 …… 134
8.6 习题 …… 136
8.7 上机题 …… 137

第 9 章 电子商务与数字现金 …… 139

9.1 安全的电子交易 …… 139
9.2 数字现金 …… 141
9.3 习题 …… 145

第 10 章 秘密共享方案 …… 146

10.1 秘密分拆 …… 146
10.2 门限方案 …… 146
10.3 习题 …… 151
10.4 上机题 …… 152

第 11 章 博弈 …… 153

11.1 电话掷币 …… 153
11.2 电话扑克 …… 155
11.3 习题 …… 158

第 12 章 零知识证明 …… 159

12.1 基本构成 …… 159
12.2 Feige-Fiat-Shamir 识别方案 …… 161
12.3 习题 …… 162

第 13 章 密钥建立协议 …… 165

13.1 密钥协商协议 …… 165
13.2 密钥预分发 …… 167
13.3 密钥分发 …… 168
13.4 公钥基础设施（PKI） …… 171
13.5 习题 …… 173

第 14 章 信息论 …… 175

14.1 概率回顾 …… 175
14.2 熵 …… 177
14.3 哈夫曼编码 …… 180
14.4 完全保密 …… 181
14.5 英文的熵 …… 183
14.6 习题 …… 187

第 15 章 椭圆曲线 …… 189

15.1 加法定律 …… 189
15.2 模 n 椭圆曲线 …… 192

15.2.1 模 p 点的数目 …… 193
15.2.2 基于椭圆曲线的离散对数 …… 193
15.2.3 表示明文 …… 194
15.3 用椭圆曲线因数分解 …… 194
15.4 特征为 2 的椭圆曲线 …… 197
15.5 椭圆曲线密码体制 …… 199
15.5.1 椭圆曲线 ElGamal 密码体制 …… 199
15.5.2 椭圆曲线 Diffie-Hellman 密钥交换 …… 200
15.5.3 ElGamal 数字签名 …… 200
15.6 习题 …… 201
15.7 上机题 …… 203

第 16 章 纠错码 …… 205

16.1 绪论 …… 205
16.2 纠错码 …… 209
16.3 一般编码的边界条件 …… 212
16.3.1 上边界条件 …… 212
16.3.2 下边界条件 …… 213
16.3.3 例子 …… 215
16.4 线性码 …… 216
16.5 汉明码 …… 221
16.6 Golay 码 …… 222
16.7 循环码 …… 228
16.8 BCH 码 …… 232
16.9 Reed-Solomon 码 …… 237
16.10 McEliece 密码体制 …… 238
16.11 其他问题 …… 240
16.12 习题 …… 241
16.13 上机题 …… 243

第 17 章 密码学中的量子技术 …… 244

17.1 一个量子实验 …… 244
17.2 量子密钥的分发 …… 246
17.3 Shor 算法 …… 248
17.3.1 因数分解 …… 249
17.3.2 离散的傅立叶变换 …… 249
17.3.3 Shor 的算法 …… 251
17.3.4 连分数 …… 254
17.3.5 结束语 …… 255

17.4 习题 …… 255

附录 A Mathematica 实例 …… 257

A.1 Mathematica 入门 …… 257
A.2 部分命令 …… 258
A.3 第 2 章实例 …… 259
A.4 第 3 章实例 …… 265
A.5 第 6 章实例 …… 267
A.6 第 8 章实例 …… 273
A.7 第 10 章实例 …… 273
A.8 第 11 章实例 …… 274
A.9 第 15 章实例 …… 275

附录 B Maple 实例 …… 279

B.1 Maple 入门 …… 279
B.2 部分命令 …… 280
B.3 第 2 章实例 …… 281
B.4 第 3 章实例 …… 286
B.5 第 6 章实例 …… 289
B.6 第 8 章实例 …… 294
B.7 第 10 章实例 …… 294
B.8 第 11 章实例 …… 295
B.9 第 15 章实例 …… 296

附录 C MATLAB 实例 …… 300

C.1 MATLAB 入门 …… 300
C.2 第 2 章实例 …… 304
C.3 第 3 章实例 …… 314
C.4 第 6 章实例 …… 317
C.5 第 8 章实例 …… 321
C.6 第 10 章实例 …… 321
C.7 第 11 章实例 …… 322
C.8 第 15 章实例 …… 324

附录 D 进一步阅读的建议 …… 330

参考文献 …… 331

第 1 章

密码学及其应用概述

人类一直以来就对保护信息以不被他人所知有强烈的兴趣，孩提时代，我们当中的很多人就梦想有一个魔幻解码环，使我们能够和朋友交换编码信息而不让父母、兄弟姐妹和老师知道。随着历史的发展，已有无数个事例使人们做到了确保秘密的信息不让敌人得知。国王或将军与他们的军队使用加密的方法来传递战时机密情报，以防止敌人窃取。事实上，据说朱丽叶斯·凯撒（Julius Caesar）发明了一种简单的密码，后来该密码就以他的名字命名。

随着社会的发展，人们对更成熟和更完善的保护数据的方法的要求越来越高，在如今的信息时代，这种呼声愈发强烈。因为世界已经相互关联在一起，对信息和电子设施的使用不断增加，由此势必带来对电子系统依赖性的增加。目前已经存在涉及一些敏感信息的交换，比如信用卡在因特网上的使用已相当普遍。有效地保护这些数据和电子系统在我们的生活中已经刻不容缓。

保护数据的方法属于密码学范畴。实际上，该学科包括 3 个名字：**密码学**（**cryptography**）、**密码术**（**cryptology**）和**密码分析学**（**cryptanalysis**），这 3 个名字经常交替使用。从技术上严格来说，密码术是研究在不安全通道传递信息及相关问题的总称，设计体制来完成这些功能的过程叫密码学，密码分析学则涉及到如何破坏这样的体制。当然，目前基本上不存在在密码学和密码分析学两个领域中都可以使用的十分完美的方法。

术语**编码理论**（**coding theory**）常用来描述密码学，但这经常会导致混淆。编码理论所提到的输入信息符号和输出信息符号叫做编码符号（code symbol）。编码理论覆盖三方面的基本应用：压缩（compression）、保密（secrecy）和纠错（error correction），在过去的几十年里，术语编码理论已几乎成为纠错码的代名词，如此一来，编码理论集中研究的是在噪声信道传输信息及如何确保收到的信息是正确的，而对密码学来说，研究的是如何保证在不安全的通道上安全传输信息。

虽然纠错码不是本书关注的重点，但在这里我们还是应该强调一下，在任何实际系统中，纠错码经常是和加密联系在一起的，因为在一个设计周到的密码体制中，哪怕是一丁点的改变也会完全破坏信息的完整性。

现代密码学领域主要依赖于数学、计算机科学和人的聪明才智。本书介绍了确保数据传输和电子系统安全的数学理论和协议以及其他一些相关技术，如数字签名（electronic signature）和秘密共享（secret sharing）等。

1.1 安全通信

基本的通信方案如图 1.1，它包括两个当事人，我们称为艾丽斯（Alice）和鲍勃（Bob），彼此要进行通信，第三方伊芙（Eve）是一个可能的偷听者。

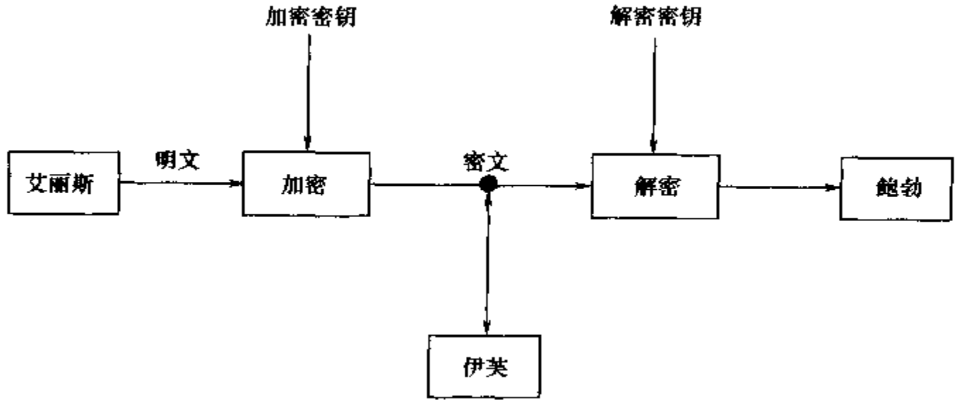


图 1.1 密码学的基本通信方案

当艾丽斯要传递一个消息（称为**明文，plaintext**）给鲍勃时，她使用事先和鲍勃约定好的方法加密要传递的消息，通常，这个加密方法假定伊芙是知道的，保持信息的机密性靠的是**密钥（key）**，当鲍勃收到了这个加过密的消息（称为**密文，ciphertext**）时，他使用解密密钥将消息翻译成明文。

伊芙可能会采取以下某种手段：

1. 读取这个消息。
2. 寻找这个密钥并读取用该密钥解密的所有消息。
3. 中断艾丽斯的消息并用同样的方法改变此消息，使鲍勃认为艾丽斯发送了变换的消息。
4. 伪装成艾丽斯去和鲍勃通信，使鲍勃以为他正和艾丽斯通信。

我们会遇到何种情况取决于伊芙的邪恶程度，步骤（3）和步骤（4）分别涉及的是完整性（integrity）和鉴别（authentication）的问题，这里简单地讨论一下。本书中把更活跃和恶毒的对手称之为马洛里（Mallory），如（3）和（4）所对应的那样，而把被动的观察者称之为奥斯卡（Oscar）如（1）和（2）对应的。我们主要讨论的是伊芙这一方，并假定她尽可能的恶毒。

1.1.1 可能的攻击

伊芙可能采用 4 种主要的攻击手段。它们的主要区别在于，当伊芙想法确定密钥时所能获取的信息数量不同。4 种攻击方式如下：

1. 仅知道密文（ciphertext only）攻击：伊芙仅能得到一份密文的拷贝。

2. 已知明文（known plaintext）攻击：伊芙不仅有一份密文的拷贝还有其对应的明文。

举个例子，假设伊芙中途截获了经加密压缩的密文，第二天又获得了经解密释放过的明文。如果她能推论出密钥，并且假如艾丽斯没有改变这个密钥，那么伊芙就可以阅读以后艾丽斯和鲍勃传递的所有消息。又假如，艾丽斯和鲍勃通信总是以“亲爱的鲍勃”开头，那么伊芙就获得了一小部分密文及其对应的明文，对于大多数较弱的密码体制，这些信息足够用来破译密钥，甚至对功能稍强些的密码体制，如第二次世界大战中德国的“Enigma”机器，这些信息也足以破译密钥了。

3. 选择明文（chosen plaintext）攻击：伊芙临时获取到了加密的机器，她不能打开它去知道密钥；但她可以通过加密大量挑选出的明文，然后试着利用其产生的密文来推测密钥。

4. 选择密文（chosen ciphertext）攻击：伊芙临时获得了用来解密的机器，利用它去“解密”几串符号，并尽可能利用结果推测出密钥。

选择明文攻击可能在如下情况中发生，当要分辨一架飞机是敌还是友时，可以发送一个随机的信息给飞机，然后飞机自动地加密该信息并将它发送回来。友机被认定有正确的密钥，可以用正确的加密信息来比较从飞机传来的信息，如果它们正确，就认为飞机是友机，如果对不上，就认为是敌机。但是，敌机可以利用这一点，发送大量的可选择消息给我方的任意一架飞机，然后查看所得到的密文，假设这样使敌机推测出了密钥，那么敌机就可以伪装成我们的友机。

第二次世界大战时撒哈拉沙漠就出现过已知明文攻击的事例，一个孤独的德国哨兵每天发送同样一条消息说在这儿百无聊赖，当然该消息是用当天的密钥加密过的，这样每天盟军都收到一对明文和密文，这对测定密钥是极为有用的。蒙哥马利将军始终密切关注着这个信息，以确保传递的消息不被遗漏。

现代密码学最重要的假设之一是**克彻霍夫原理（Kerckhoffs's Principle）**：即在评定一个密码体制的安全性时，人们假定攻击方知道所有目前已使用的密码学方法。关于该原理在1883年奥古斯特·克彻霍夫（Auguste Kerckhoffs）著名的著作《La Cryptographie Militaire》中有详细论述。攻击方可以通过很多方法获得这些信息。例如，能够捕获和分析加密和解密设备，自己人也可能被发现或逮捕。因此，体制的安全性应该建立在密钥的基础之上，而不是依赖于算法的隐藏。所以我们假定伊芙知道目前已应用的所有加密的运算规则。

1.1.2 对称和公开密钥算法

加密和解密方法分为两类：**对称密钥（symmetric key）**和**公开密钥（public key）**。在对称密钥算法中，艾丽斯和鲍勃双方均知道加密和解密密钥。举一个例子来说，如果双方都知加密密钥，那么很容易从它推出解密密钥。在很多情况下加密密钥和解密密钥是相同的。古典的加密体制（1970年以前）以及最近的数据加密标准（DES）和Rijndael（AES）都是基于对称密钥的。

公开密钥是在20世纪70年代才开始出现，它对传统的密码学提出了根本性的改变。设想艾丽斯希望和鲍勃安全地通信，但是他们相隔上百公里，根本不可能就使用同一个密钥达成一致，看起来似乎也不大可能事先双方约定一个密钥，或委派一个值得信赖的人去传递这

个密钥，当然艾丽斯不能利用公共渠道去传递这个密钥给鲍勃，再利用这个密钥加密消息。令人烦恼的事情终于有了解决办法，即公开密钥加密法。这个密钥是公开的，并且除了鲍勃知道，几乎不可通过计算来发现解密密钥。最流行的算法是 RSA（见第 6 章），它主要基于对大整数因数分解的困难性上，其他的一些算法（见第 7 章和第 16 章）主要有 ElGamal（基于离散对数问题）和 McEliece（基于纠错码）。

这儿用一个非数学的方式来解释公开密钥算法。鲍勃给艾丽斯一个未锁上锁的盒子，艾丽斯将信息放人盒子，用这个锁锁上盒子，然后传给鲍勃，当然只有鲍勃能打开这个盒子并读取信息，先前所提到的公开密钥的方法在数学上就是和该思想联系起来的。很明显，这儿有很多确认的问题需要解决，例如伊芙可能截获第一次的传递，然后换上她自己的锁，这样当艾丽斯传给鲍勃的盒子被伊芙截获后，她就能很轻易地解开锁并读取该信息。这是使用公开密钥体制必然涉及到的一般性问题。

公开密钥加密体制所代表的很可能是密码学有趣的发展历史的最后一步，在早期的密码学中，安全性依赖于加密算法的保密程度，后来假定这些算法已为人所知，加密体制的安全性则依赖于（对称）密钥的保密和不公开。在公开密钥密码学中，算法和加密密钥是公开的，每个人都知道必须做的是寻找解密密钥，这里的安全性依赖（或希望）的是不可能通过计算获取解密密钥。看起来相当荒谬的是，一方面多年来加密算法大量出现，而另一方面对手也获取了更多关于这些算法的信息。

公开密钥方法功能相当强大，似乎它的强大使对称密钥方法成为过时，但是，它所带来的是不自由和计算的代价，在公开密钥算法中所需要的计算量比一般的加密算法如 DES 或 Rijndael 计算量多几个幂的数量级。一个重要的原则是公开密钥不要使用在数据量相当大的加密中，基于此原因，公开密钥一般应用在仅有少量数据需要传输的情况下（如数字签名、传送对称密钥算法中的密钥等）。

在对称密钥密码学中，有两种类型的密码：连续密码（stream cipher）和分组密码（block cipher）。连续密码中，输人到算法中的数据是小的段（比特或字节），而输出结果是其相对应的小段；而分组密码中，输人的是一组比特，相应的输出也是一组比特流。在 2.11 节中我们将讨论关于连续密码的例子，即线性反馈移位寄存器（linear feedback shift register）。我们关注的绝大部分情况是分组密码，为此将介绍两个非常重要的例子，第一个是 DES，第二个是 Rijndael，它在 2000 年被国际标准和技术委员会选中以取代 DES。公开密钥方法如 RSA 也被认为是分组密码。

最后，我们介绍一下两种不同类型的加密即**编码（code）**和**密码（cipher）**在历史上的差别，对于编码，它可用码字（codeword）代替（可以是一串符号），如第一次世界大战中英国海军使用 03680C，36276C 和 50302C 来代表“shipped at”、“shipped by”和“shipped from”。编码有对那些无明确意义的单词无法使用的缺点。反之对于密码来说，不必担心消息语言上的结构，而可以加密每一串字符，不管它代表的是有意义或没意义还是其他什么，因而密码比编码更通用。在早期的密码学中经常使用编码一词，有时相对应地使用密码，直到今天仍然还是这样使用，隐蔽的行动经常用密码，但无论如何，任何秘密要想安全传输都需要用密码加密。在本书中，我们一律使用密码。

1.1.3 密钥长度

衡量加密算法的安全性是一件不容易的事情，大多数算法使用了密钥，算法的安全性就和破解密钥的难易程度是紧密相关的，最简单的破解密钥的方法就是去尝试各种可能的密钥，看哪一个更接近实际的解密，这种类型的攻击称之为**蛮力攻击（brute force attack）**。在蛮力攻击类型中，密钥的长度与将要检索的整个密钥空间紧密相关，比如，如果密钥是16个比特长，那么就有 $2^{16}=65536$ 种可能的密钥。在DES算法中密钥长56比特，这样就有 $2^{56}\approx 7.2\times 10^{16}$ 种可能的密钥。在本书中，很多情况下我们将碰到类似的情况，似乎通过尝试所有可能的密钥，某个密码体制就可以轻易被破解，但说起来容易做起来难。设想你必须尝试 10^{30} 种可能性，并且计算机能够具有每秒 10^9 次的运算速度，而一年有 3×10^7s，这样完成计算需花费比 3×10^{13} 年还要多的时间，这比我们所预测的宇宙寿命还要长。

长的密钥尽管有很多优点，但它并不足以保证攻击者不能破解这个密钥，其实算法本身也起着很重要的作用。利用蛮力攻击以外的方法，一些算法也可能被攻破，另外还要牢记的是某些算法并不能很有效地利用它们的密钥长度，并不是所有具有128比特的密钥算法效率都相同。

举个例子来说，最容易被攻破的密码体制之一是替换密码，在2.4节我们将进行讨论。在这种密码体制中，可能的密钥数量是 $26!\approx 4\times 10^{26}$，与此相反，DES算法（见第4章）仅有 $2^{56}\approx 7.2\times 10^{16}$ 个密钥，但在特制的计算机上运算一天才能发现一个DES密钥，两种体制的区别就在于替换算法的攻击者使用了该算法潜在的结构，而攻击DES只能通过蛮力攻击方式，尝试所有可能的密钥来达到目的。

蛮力攻击应该是最后的攻击方式，密码专家总是希望尽快地寻找到破解方法，如可以采用频率分析（对应替换和Vigenère密码）和生日攻击（对应离散对数）。

还要提醒读者注意的是有些算法现在看起来是安全的，但并不意味着它将来也是安全的，人类的智慧在密码学协议上已经产生过许多创造性的突破，在现代密码学中有许多的例子说明，一个算法或协议被成功破解就是由于在糟糕的实施中出现了一点漏洞，或者由于技术上的进步。DES算法经受住了密码学家20年的详细审查，但最终还是被一台著名的并行计算机成功攻破，甚至当你阅读这本书的时候，量子计算机的研究正在进行中，它的出现可能会戏剧性地改变密码算法的未来。

例如，我们在后续章节准备研究的几个密码体制，其安全性主要依赖于对大整数因数分解的困难程度，如200位的数，设想要分解这么大的数 n，传统的方法是将 n 除以所有的素数直到 n 的平方根中止，小于 10^{100} 就大约有 4×10^{97} 个素数，尝试完所有的素数是不太可能的。宇宙中电子的数量估计略少于 10^{90}，输入计算后不久，电子计算机会发出请求停止的信息，很显然，需要采取更高级的的因数分解算法来取代蛮力的攻击方式。在RSA发明的时候，有一些好的因数分解算法可以使用，但也有人预言，一个长129位的数字如RSA中这个具有挑战性的数字（见6.5节）在可预测的将来是不可能被分解的，然而，先进的算法和计算机体系结构已经使这样的因数分解变得很程序化（虽然需要相当大的计算资源），因此目前来看几百位的数字长度才被认为是安全的。但假如相当规模的量子计算机出现后，那么即使是上述长度的因数分解也会变得很容易，因而整个RSA方案（包括其他许多方法）

都将需要重新考虑。

自然有人会问这样一个问题，是否有不能被攻破的密码体制，而又为什么没被采用呢？

答案是肯定的，有一种体制被称之为一次一密（*one-time pad*），它是不可攻破的，即使用蛮力的攻击方式也不行，但不幸的是使用一次一密是无穷的，它要求密钥和明文一样长，而且密钥只能使用一次，因此，对算法的一种选择是使用适当的密钥长度保证算法实施正确，这样在合理的时间内不可攻破。

在考虑密钥长度时很重要的一点是，许多情况下，人们可以通过稍微增加密钥的长度来从数学上增加算法的安全性，但这并不总是可行的。如果正工作在使用芯片的计算机下，它可以处理64比特的字长，若把密钥的长度从64比特增加到65比特，可能就要考虑硬件是否可行了，它可能相当昂贵。因此，好的加密体制方案涉及数学和工程设计两个方面。

最后，我们还需要了解一下关于数字大小的一些术语。直觉可能会告诉你处理20位数字的花费是处理10位数字的2倍，在某些算法下是正确的。然而，如果将位数提升到10^{10}，未达到10^{20}；处理10位的花费就仅是这种方式的100亿分之一。类似地，对60比特密钥所采用的蛮力攻击要比对30比特密钥所花费的代价高10亿倍。

有两种方式来衡量数字的大小：一是数字n的实际大小，二是它的十进制表示的位数（也可使用二进制来表示），这接近于$\log_{10}(n)$。利用传统的方法，一位数通过乘法乘到k位数n，其花费是k^2或近似于（$\log_{10}n$）^{2}。通过除法分解数n需要除以所有的素数直至n的平方根，花费大约是$n^{1/2}$。算法采用$\log n$的幂比直接用n的幂更理想。就本例来说，如果将n的位数加倍，利用因数4的因数分解所花费的时间增加到n的平方，而利用因数n的分解增加花费的时间则是无法计算的。当然也有一些好的算法可以兼顾这些操作的两个方面，但目前来看，因数分解比乘法产生的作用更大。

接下来我们还将看到某些算法花费$\log n$的时间来进行某些计算（例如，寻找最大公约数和模数的求幂）。另外还有其他一些众所周知的算法，它们的运算效果只比n的幂次方法稍好一些（例如，因数分解和寻找离散对数）。最快与最慢算法之间的相互影响是本书中将要讨论的密码学算法的基本依据。

1.2 密码学应用

密码体制不仅涉及到加密和解密消息，还涉及到解决现实世界对信息安全的要求问题。这里提出4个主要的目标：

1. 机密性（confidentiality）：伊芙不能读取艾丽斯发给鲍勃的信息，主要的手段就是加密和解密算法。

2. 数据的完整性（data integrity）：鲍勃需要确认艾丽斯的消息没有被改过。例如，可能产生传输错误，也有可能攻击者截获这个消息，并在到达目的地之前修改过它。密码学领域有许多原始的方法，如散列函数，它提供了检测数据是否被攻击者有意或无意地修改过的方法。

3. 鉴别（authentication）：鲍勃需要确认他接收到的信息仅是艾丽斯发送的，在此前提下，也包括身份识别和口令确认（这种情况下，鲍勃就指的是计算机）。在密码学中实际上

提出了两种类型的鉴别：实体鉴别（entity authentication）和数据源发鉴别（data-origin authentication）。所说的术语识别（identification）常用来表示实体鉴别，它主要关注的是传输中所涉及的人员身份的鉴别，数据源发鉴别集中关注数据最初所涉及的信息，如数据的创造者和生成时间等。

4. 反拒绝（non-repudiation）：艾丽斯不能声称她没有发送过消息，反拒绝在电子商务应用中是特别重要的，因为消费者不能否认他对商品的实际消费行为。

鉴别和反拒绝在概念上是很相近的，但有一点不同。在对称密钥加密体制中，鲍勃可以确认消息来自于艾丽斯（排除有人知道艾丽斯的密钥），因为没有人能够加密鲍勃成功解密过的消息，因此鉴别是自动进行的，但这并不能证明一定是艾丽斯发过来的消息，因为也可能是鲍勃自己给自己发送的消息，可见反拒绝基本上是不可能的。在公开密钥加密体制中，鉴别和反拒绝两方面都能够得到实现（见 6.7 节和第 8 章）。

本书的很多地方会讲到一些特定的密码学应用，在正文和练习中都有。下面是基本的概述。

数字签名（digital signature）：纸张和墨水最重要的特点之一是签名，通过签署一个文件，这个人的身份就和这条信息联系起来了，假设一个人伪造其他人的签名来签署文件是相当困难的。反过来，电子信息是很容易被精确地拷贝的，那么，我们又如何阻止敌人截取一个文件上的签名，利用它去伪造到另一个文件上呢？所以我们就要研究密码协议，它允许电子信息在这样一种情况下被签署，即每个人都相信这个签名人就是签署文件的人，而且这个签名人还不能否认他签过的文件。

身份识别（identification）：当要登录机器或建立最初的传输连接时，用户需要确认他或她的身份，简单地写入用户名是不够的，也不能证明用户真实的身份，典型的方法是采用口令机制。我们将涉及到各种各样确认身份的方法。在 DES 一章中要讨论口令文件，后面的章节中还提出了 Feige-Fiat-Shamir 鉴别方案，它是一种在不知口令的情况下来证明身份的“零知识基础方法”。

密钥确立（establishment）：当大量的数据需要加密的时候，最好采用对称密钥加密算法，但当艾丽斯没有机会遇到鲍勃时，艾丽斯如何将密钥给鲍勃呢？有多种方法，一种方法就是采用公开密钥，另外就是 Diffie-Hellman 密钥交换算法，还有一种不同的方法就是由一个可信赖的第三方去给艾丽斯和鲍勃密钥。两个事例就是 Blom 的密钥生成方案和 Kerberos 算法，它们都是非常流行的对称密钥加密协议，在网络用户之间交换密钥过程中提供了鉴别和安全性。

秘密共享（secret sharing）：第 10 章我们将介绍秘密共享的方案，假如你在银行中存放了保险箱，但你并不相信任何个人去打开保险箱，进一步说，你划分一组人可打开保险箱，但至少这些人中的两个人同时在场时才可打开保险箱，秘密共享就可以解决这个问题。

电子商务（E-commerce）：我们怎样才能在诸如 Internet 这样的通道上实现安全的交易呢？又如何防止利用信用卡进行欺诈交易呢？我们要讨论如何使用双重签名。

电子货币（electronic cash）：信用卡及类似的设备是方便的，但它不提供匿名，显然电子兑付是很有用的，至少对一些人是这样。但是，电子实体可以被拷贝，有事例说明某电子兑付系统提供了匿名机制由此抓住了仿造者。

博弈（game）：如何同与你不在一个房间的人玩掷硬币和扑克游戏呢？比如发牌就是问题，我们将讲解如何利用密码学的思想来解决这些问题。

第2章 古典密码体制

纵观历史，使信息让对手难以破解的方法是相当重要的，本章我们将介绍在计算机发明以前常使用的一些经典的密码体制。尽管从现在来看，特别是用计算机来处理的话，这些密码体制相当脆弱，以致于如今已不再使用，但是它们关于密码学的重要思想给了后人很好的启示。

首先，针对这些简单的密码体制，我们有如下约定：

- 明文用小写字母表示，密文则用大写字母表示（在上机题部分中例外）。
- 英文字母表和如下数字对应：

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

q	r	s	t	u	v	w	x	y	z
16	17	18	19	20	21	22	23	24	25

注意，这里从 $a=0$ 开始，因此 z 代表数字 25。因为很多人习惯于用 1 表示 a，26 表示 z，上述这种表示方式你可能不太习惯，但它表征了后面要介绍的基本的密码体制。

- 忽略空格和标点符号，这可能让人不太适应，但是经过解密后，几乎总是可以正确地还原明文中的这些空缺。如果保留这些空缺，有两种情况：一种情况就是在加密时将明文中的这些空缺仍然按原样保留，这样的话会在信息的结构中产生很多信息使解密变得容易；另一种情况是将空缺也加密，但在密文中会有很多相同的信息频繁出现（除非信息很平均，即每个单词至少有 8 个字母），同样使解密变得容易。

注：本章我们将使用数论方面的一些概念，特别是模运算方法，如果这部分内容你不熟悉的话，在开始阅读下面章节之前，请先阅读第 3 章的前 3 节。

2.1 移位密码

早期的密码体制创始人之一是朱丽叶斯·凯撒（Julius Caesar）。假设他要发送如下明文信息：

gaul is divided into three parts

但他并不想让布鲁吐斯（Brutus）读到它，于是他将每一个字母向后移三位，这样 a 变成 D，b 变成 E，c 变成 F 等等，字母表中最后的字母又递归转到开始，即 x 变成 A，y 变成 B，z 变成 C，如此一来，密文就成为：

JDXOLVGLYLGHGLQWRWKUHHSDUWV

解密的过程就是将字母移回三位即可（并尽量判断一下怎样将空格还原）。

现在我们给出一般形式。如果你不熟悉模运算，在继续学习下面的内容之前请阅读第 3 章前几节。

字母表用整数 0～25 来标识，密钥是一个整数 k，取值是 $0 \leqslant k \leqslant 25$，加密过程如下：

$$x \mapsto x + k \pmod{26}。$$

解密过程是 $x \mapsto x - k \pmod{26}$，例如，凯撒令 $k = 3$。

让我们来看看 4 种类型的攻击是怎样工作的：

1. 仅知道密文（ciphertext only）攻击：伊芙仅仅有密文，她最好的策略是穷尽检索，因为只有 26 种可能的密钥。如果该信息比某些字母长（后面讨论到熵时将有更精确的描述），那么就不可能有多于一种意义的信息，这就是明文。如果你不相信的话，试着去发现出 4 或 5 个字母组成的一些单词它们相互移位的情况，在练习题 1 中有一个这样的示例。另外一种可能的情况是，这个信息相当长并且频繁地出现各种各样的字母，那么字母 e 在大多数英文文本中出现的频率是最高的，假设字母 L 在密文中出现的频率最高，因为 $e = 4, L = 11$，一个合理的猜测是 $k = 11 - 4 = 7$，但在移位密码中这种方法比穷尽搜索法要花费更长的时间，加之该方法为了能正确工作，要求在信息中有更多的字母（无论多短，如 this，不包括一个相同的符号，否则就改变了统计的计算值）。

2. 已知明文（known plaintext）攻击：如果仅知道与密文相对应的明文的一个字母，就可推断出这个密钥值。例如：如果已知 $t(=9)$ 加密成了 $D(=3)$，那么该密钥是 $k \equiv 3 - 19 \equiv -16 \equiv 10 \pmod{26}$。

3. 选择明文（chosen plaintext）攻击：选择字母 a 作为明文，通过密文就能得到这个密钥。例如，如果密文是 H，那么密钥值是 7。

4. 选择密文（chosen ciphertext）攻击：选择字母 A 作为密文，从明文中可知密钥值。例如，如果明文是 h，那么密钥就是 $-7 \equiv 19 \pmod{26}$。

2.2 仿射密码

通过如上说明，能够看到移位密码不能很好地适应特殊环境，并且它的抗破译性也不是很强壮。设两个整数 α 和 β，及 $\gcd(\alpha, 26) = 1$，考虑这样的函数（称之为仿射函数（affine function））

$$x \mapsto \alpha x + \beta \pmod{26}。$$

假设 $\alpha = 9$ 和 $\beta = 2$，这样可得出 $9x + 2$，取一个明文字母如 $h(=7)$，它加密成 $9 \cdot 7 + 2 \equiv 65 \equiv 13 \pmod{26}$，就是字母 N。使用同样的函数我们得到

$$affine \mapsto CVVWPM。$$

怎样来解密呢？如果我们利用有理数而不是模 26，可从 $y = 9x + 2$ 中解出：$x = \frac{1}{9}(y - 2)$，但当用模 26 来计算时，$\frac{1}{9}$ 需要重新解释，因为 $\gcd(9, 26) = 1$，这是一个对 9（mod 26）的乘法逆元素（如果这句话不明白，请阅读 3.3 节）。事实上 $9 \cdot 3 \equiv 1 \pmod{26}$，因此

3 就是要得到的乘法逆元素，并且可以替代 $\frac{1}{9}$，因此有

$$x \equiv 3(y-2) \equiv 3y-6 \equiv 3y+20 \pmod{26}。$$

来测试一下，字母 $V(=21)$ 被映射成 $3 \cdot 21+20 \equiv 83 \equiv 5 \pmod{26}$，这是字母 *f*。类似地，我们看到密文 *CVVWPM* 的明文是 *affine*。

设想我们尽量用函数 $13x+4$ 作为加密函数，可得

$$input \mapsto ERRER。$$

如果修改这个输入，可得

$$alter \mapsto ERRER。$$

很明显该函数出错了，它不可能用来解密，因为几个明文产生了相同的密文，特别要说明的是，加密必须是一对一的，上述例子是个失败。

该例什么地方出错了呢？如果解 $y=13x+4$，可得 $x=\frac{1}{13}(y-4)$，但 $\frac{1}{13}$ 模 26 的结果不存在，因为 $\gcd(13,26)=13 \neq 1$。一般情况下的表达式是，$\alpha x+\beta$ 是模 26 一对一的函数，并且仅有一个解使 $\gcd(\alpha,26)=1$，这样利用 $x \equiv \alpha^{*}y-\alpha^{*}\beta \pmod{26}$ 解密，这儿 $\alpha\alpha^{*} \equiv 1 \pmod{26}$，如此一来，通过仿射函数就完成了解密。

这种加密方法的密钥就是一对 (α,β)，对 $\gcd(\alpha,26)=1$ 中的 α 有 12 种可能的选择，对 β 有 26 种选择（因为用 mod 26 来计算，所以仅需要考虑介于 0～25 之间的 α 和 β）。因此，密钥值一共有 $12 \cdot 26=312$ 种选择。

下面我们看一下可能的攻击。

1. 仅知道密文攻击：穷尽搜索覆盖所有 312 个密钥，虽然该方法比同样方法的移位密码花费的时间长，但在计算机中计算起来非常容易，当所有密钥的可能值试过以后，一段相当短的明文，一般是 20 个字符左右，就可能推算出密钥，并且与密文相对应的仅有惟一的明文，这样就能确定密钥的值。另一种方法是采用频率计算，这样可能要求有更长的明文。

2. 已知明文攻击：如果运气好的话，知道了明文的两个字母及其对应的密文内容，就能够找出密钥。即便未能找出密钥，密钥取值的范围大大减少了，而且只需再有几个字母就可以计算出密钥。

例如，假设明文用 *if* 开头，其相应的密文是 *PQ*，用数字表示，意味着 $8(=i)$ 映射成 $15(=P)$，5 映射成 16，因此有如下等式：

$$8\alpha+\beta \equiv 15 \text{ 和 } 5\alpha+\beta \equiv 16 \pmod{26}。$$

两式相减，得出 $3\alpha \equiv -1 \equiv 25 \pmod{26}$，可得出惟一的解 $\alpha=17$。使用第一个等式，有等式 $8 \cdot 17+\beta \equiv 15 \pmod{26}$，求出 $\beta=9$。

假设明文 *go* 相对应的密文是 *TH*，我们可以得到如下等式：

$$6\alpha+\beta \equiv 19 \text{ 和 } 14\alpha+\beta \equiv 7 \pmod{26}。$$

两式相减，得出 $-8\alpha \equiv 12 \pmod{26}$，由于 $\gcd(-8,26)=2$，于是得出两个解 $\alpha=5,18$。相应的 β 值都是 15（这并非巧合，而总是如此），这样有两个候选密钥：(5，15) 和 (18，15)，但 $\gcd(18,26) \neq 1$，于是去掉第二个，因此密钥是 (5，15)。

前面所介绍的计算过程中排除了 gcd 是 13（或 26）的情况，如果有这种情况出现，那么在可能的情况下使用信息中的其他字母来计算。

假如人们仅知道明文中的一个字母，仍然可能得到介于 α 与 β 的关系。例如，已知明文 g 相应的密文是 T，就有 $6\alpha+\beta=19(\bmod 26)$，就有 12 种可能的 α，而每一种又对应一个 β 值，因此，穷尽搜索 12 个密钥就可以计算出正确的密钥。

3. 选择明文攻击： 选择 ab 作为明文，那么密文的第一个字母将是 $\alpha\cdot 0+\beta=\beta$，第二个字母将是 $\alpha+\beta$，因此可以找到密钥。

4. 选择密文攻击： 选择 AB 作为密文，求出解密的函数形式是 $x=\alpha_1 y+\beta_1$，求解 y 就得到加密密钥。但令人烦恼的是，虽然能得出解密函数，但到底选择哪一个才是所需要的？

2.3 Vigenère 密码

各种各样的移位密码是在 16 世纪发明的，但它们大多数来自于 Vigenère 方法，尽管 Vigenère 加密方法更复杂和高级。直到如今的 20 世纪，这种加密体制在很多地方被认为是安全的，虽然早在 19 世纪，Babbage 和 Kasiski 就已展示了如何攻击它们。在 1920 年，Friedman 开发了另外一些加密方法，打破了 Vigenère 及其相关的密码方法。

这个加密的密钥是一个向量，按如下方式来选择。首先，确定一个密钥长度，如 6，然后从 0～25 个整数中选择元素项满足这个长度的向量，如 $k=(21,4,2,19,14,17)$。通常情况下密钥所对应的单词是很容易记忆的，在这里就称这个单词为向量。系统的安全性所依赖的就是既不能知道密钥内容也不能得知其长度。

下面所举的例子就是利用 k 来加密信息，首先，取明文的第 1 个字母并将之移 21 位，然后将第 2 个字母移 4 位，第 3 个字母移 2 位等等，一旦到了密钥的末尾，又从头开始，这样第 7 个字母又移 21 位，第 8 个字母移 4 位等等，加密过程的密码流程表如下：

（明文）	*h*	*e*	*r*	*e*	*i*	*s*	*h*	*o*	*w*	*i*	*t*	*w*	*o*	*r*	*k*	*s*
（密钥）	21	4	2	19	14	17	21	4	2	19	14	17	21	4	2	19
（密文）	*C*	*I*	*T*	*X*	*W*	*J*	*C*	*S*	*Y*	*B*	*H*	*N*	*J*	*V*	*M*	*L*

如果知道足够的字母的话，针对已知明文的攻击方式就能成功，因为这里的密钥简单地通过从密文模 26 再减去明文就可以求出。对可选择的明文攻击这种情况，利用明文 *aaaaa*…… 马上可以求出密钥，然而，对可选择的密文攻击用 *AAAAA*…… 将产生负的密钥。但是假设你仅有密文，针对仅知密文的攻击，一直以来认为采用这种方法是比较安全的，但现在对这种情况也是很容易找到密钥的。

密码分析所基于的事实是在绝大多数英文文本中，字母的频率是不相等的。比如，e 比 x 出现得更频繁。这些字母的频率在［Beker-Piper］中已经制成了表格，在此提供在表 2.1 中。

表 2.1 英文字母频率表

a	b	c	d	e	f	g	h	i	j
.082	.015	.028	.043	.127	.022	.020	.061	.070	.002
k	l	m	n	o	p	q	r	s	t
.008	.040	.024	.067	.075	.019	.001	.060	.063	.091
u	v	w	x	y	z				
.028	.010	.023	.001	.020	.001				

当然，这也不是一成不变的，虽然这是基于普遍情况得出来的统计值。有一本由 Ernest Vincent Wright 写的书 *Gadsby* 中就不包括一个字母 *e*。甚至更极端的是，在由 George Perec 用法文所写的 *La Disparition* 一书中也没有一个字母 *e* 出现（不仅未使用动词等一般形式，而且也未用几乎所有的阴性名词和形容词）。有一本由 Gilbert Adair 翻译的英文书 *A Void*，也未用到一个字母 *e*。但一般来说，只要有几百个字母组成的文本，我们仍假设上述表所给出的大略估计总是正确的。

如果我们使用简单的移位密码，那么对字母 *e*，举个例子来说，在密文中将总是以某个字母出现，它与字母 *e* 在明文中有相同的出现频率，因此，通过频率分析就能解密出这个字母。但是这个例子用在前面的 Vigenère 密码中，字母 *e* 很可能作为两个字母 *I* 或 *X* 出现。如果使用长一点的明文，*e* 很可能作为 *Z*，*I*，*G*，*X*，*S* 和 *V* 中的某一个值出现，对应的移位是 21，4，2，19，14，17。而密文中的 *Z* 并不仅是由 *e* 加密产生的，字母 *v* 也可能加密成 *Z*，它是文本中的字母通过移动 4 位得来的。类似地，*x*，*g*，*l* 和 *i* 都可以加密成 *Z*。因此 *Z* 的频率是由 *e*，*v*，*x*，*g*，*l* 和 *i* 在明文中的频率组合而成的，所以很难用频率数的结果去推断来自于什么字母。实际上，频率分析总是有出入的，密文中每个字母的频率更接近于 1/26。至少它们更接近于英文字母通常的分布。

这儿有一个更具说服力的事例，密文如下：

```
VVHQWVVRHMUSGJGTHKIHTSSEJCHLSFCBGVWCRLRYQTFSVGAHW
KCUHWAUVMERZHMFVVHIPVFHQWCBGELVVHWSOSWMEGWPIGUGLF
DGCIXJCBVPSVWRFWBOIKUHKICGMLWCPCZRLJSHEKGKLXZYVLG
ZVVHOWAADCTGQKEFISGMKOQSLJSUTGKBWMFHOYSJQTWLWCRRT
LKCQSXVVLWUQRHFQVVRWWFSVMJKBKLQHUEFUALXAODRVLCBWQ
WUGDKWUKLXZQIWXZGGOMYJHHWLFOQKWTCIXJSLVEGGVEYGGEI
APUUISFPBTGNWWMUCZRVTWGLRWUGUMNCZVILE
```

其字母出现的频数如下：

A	B	C	D	E	F	G	H	I	J	K	L	M
8	5	12	4	15	10	27	16	13	14	17	25	7

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
7	5	9	14	17	24	8	12	22	22	5	8	5

注意，这里没有一个字母的频数比其他大许多，正如前面所讨论的那样，这是因为 *e* 在加密的过程中扩散成了几个字母的缘故。

那么如何解密这些信息呢？有两个步骤：找到密钥的长度和发现密钥。下面我们首先讲述怎样找到密钥的长度，并给出一种发现密钥的方法，随后解释用这种方法发现密钥的工作过程，最后给出了发现密钥的第二种方法。

2.3.1 发现密钥长度

在一张长纸条上写上密文，再在另一张长纸条上写上同样的密文。将一张纸条放在另一张纸条上面，但是将其中一张纸条移动某个位置（这就是潜在的密钥长度）。比如，移动位

置后有如下两个字条：

		V	V	H	Q	W	V	V	R	H	M	U	S	G	J	G
V	V	H	Q	W	V	V	R	H	M	U	S	G	J	G	T	H
														*		
T	H	K	I	H	T	S	S	E	J	C	H	L	S	F	C	B
K	I	H	T	S	S	E	J	C	H	L	S	F	C	B	G	V
G	V	W	C	R	L	R	Y	Q	T	F	S	V	G	A	H	...
W	C	R	L	R	Y	Q	T	F	S	V	G	A	H	W	K	...
				*												

两个相同的字母下面用*标记，并数一下标记的个数，上述的文本中正好有两个，如果我们继续完成整个文本，可以数到 14 个相同的，如果采用不同的移位值，可以得到下面的数据：

移位值：	1	2	3	4	5	6
相同数：	14	14	16	14	24	12

移动 5 个位置具有最多的相同数，这个移位值就是最可能的密钥长度值，后面我们会解释。这种方法非常快，甚至不用计算机就总是可以得出密钥的长度。

2.3.2　发现密钥：第一种方法

现在假设我们确定了本例中密钥的长度是 5，看一下第 1、第 6、第 11 个……这些字母，再看哪一个字母出现的频数最高，可以得出：

A	B	C	D	E	F	G	H	I	J	K	L	M
0	0	7	1	1	2	9	0	1	8	8	0	0

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
3	0	4	5	2	0	3	6	5	1	0	1	0

这里频数最高的字母是 G，紧随其后的是 J，K，C。若 $J=e$ 表明移动了 5 位，因此 $C=x$，但这有可能在密文中对 x 产生相当高的频率值。类似地，$K=e$ 意味着 $P=j$ 和 $Q=k$，两个字母都具有相当高的频率，最后，$C=e$ 表示要求 $V=x$，这是不可能的事情。因此，我们能够确定 $G=e$ 且密钥的第 1 个元素项是 $2=e$。

现在我们来看一下第 2、第 7、第 12 个……字母，可以发现 G 出现了 10 次，S 出现了 12 次，其他字母排在后面。如果 $G=e$，那么 $S=q$，它在明文中不可能出现了 12 次，于是 $S=e$ 且密钥的第 2 个元素项是 $14=o$。

现在来看第 3、第 8、第 13 个……字母组，其频数是

A	B	C	D	E	F	G	H	I	J	K	L	M
0	1	0	3	3	1	3	5	1	0	4	10	0

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
2	1	2	3	5	3	0	2	8	7	1	0	1

最初的猜测是 $L=e$ 出现了问题；比如，$R=k$ 和 $E=x$ 有太高的频率，而 $A=t$ 又有太低的频率。类似地，$V=e$ 和 $W=e$ 似乎不太可能，最好的选择就是 $H=e$，于是密钥的第 3 个

元素项是 $3=d$。

从第4、第9、第14个……这一组字母中得出 $4=e$，作为密钥的第4个元素项。最后，第5、第10、第15个……字母组产生 $18=s$，作为密钥的最后一个元素项。因此对密钥的猜测就是

$$\{2,14,3,4,18\} = \{c,o,d,e,s\}。$$

当我们看第3、第8、第13个……这组字母时（当然也可以是第5、第10、第15个……这种情况），如果我们每次拿50个字母，就有了一组相对小一点的字母样本，在其上进行频率统计。其他的字母也可以在同样小的样本上像 e 一样进行分析。而在一般情况下，大多数高频率的字母出现的频数高，低频率的字母出现的频数低，因此在这样的前提下，此方法总是能很有效地确定与密文相对应的加密密钥。

一旦潜在的密钥找到了，通过它解密密文来测试是否正确，很容易判断出它的正确性。

在本例中，密钥应该是（2，14，3，4，18），如果使用该密钥解密这段密文，可得：

themethodusedforthepreparationandreadingofcodemessagesis
simpleintheextremeandatthesametimeimpossibleoftranslatio
nunlessthekeyisknowntheeasewithwhichthekeymaybechangedis
anotherpointinfavorofheadoptionofthiscodebythosedesirin
gtotransmitimportantmessageswithouttheslightestdangeroft
heirmessagesbeingreadbypoliticalorbusinessrivalsetc

这段文字源自于美国科学家的短文 supplement LXXXIII（1/27/1917），第61页。下面简短地解释一下前文所提到的 Vigenère 密码安全性的问题。

在阐述发现密钥的第二种方法之前，先解释一下为什么用前述的方法能找出密钥的长度。

将英语字母的频率转换成向量：

$$\mathbf{A}_0 = (.082,.015,.028,...,.020,.001)。$$

令 $\mathbf{A}_i$ 是 $\mathbf{A}_0$ 右移 i 个位置的结果，例如，

$$\mathbf{A}_2 = (.020,.001,.082,.015,...)。$$

向量 $\mathbf{A}_0$ 自身的点积是

$$\mathbf{A}_0 \cdot \mathbf{A}_0 = (.082)^2 + (.015)^2 + \cdots = .066。$$

当然 $\mathbf{A}_i \cdot \mathbf{A}_i$ 也等于 .066，因为可以得到同样的向量和，只不过最初使用的是不同的符号罢了。但当 $i \neq j$ 时 $\mathbf{A}_i \cdot \mathbf{A}_j$ 的点积是很低的，范围是从 .031 到 .045：

$\|i-j\|$	0	1	2	3	4	5	6
$\mathbf{A}_i \cdot \mathbf{A}_j$	.066	.039	.032	.034	.044	.033	.036
	7	8	9	10	11	12	13
	.039	.034	.034	.038	.045	.039	.042

点积仅依赖于 $|i-j|$，从下面的讲述中可以看出，每个向量的内容不过是向量 $\mathbf{A}_0$ 经过移位得到的。在点积运算中，$\mathbf{A}_0$ 的第 i 项与第 j 项相乘，第 $(i+1)$ 项与第 $(j+1)$ 项相乘等等，因此每一个元素项是由它自己和第 $j-i$ 位置的元素相乘而得来的。于是点积仅依赖于 $i-j$ 的不同。但是若将 i 和 j 交换位置，注意到 $\mathbf{A}_i \cdot \mathbf{A}_j = \mathbf{A}_j \cdot \mathbf{A}_i$，可见 $i-j$ 和 $j-i$ 得出了相同的点积，因此点积仅仅依赖于 $|i-j|$，在上述表中，仅需要计算到 $|i-j|=13$ 即可，比如，$i-j=17$ 相对应的是在一个方向上移位17次，或在另一个方向上移位9次，因此利用 $i-j=9$

能得出同样的点积。

点积 $\mathbf{A}_0 \cdot \mathbf{A}_0$的结果比其他的点积结果大，原因是在向量中大的数是成对出现的，小的数也成对出现，而在其他的点积中，大数和某个其他的数成对出现，这就影响了它们的结果。

假定在明文中字母的分布非常接近于英文字母的分布规律，正如前面向量 $\mathbf{A}_0$所表示的那样。查看上面纸条中任意的密文字母，可以看到它是由某个英文字母移动 i 位（对应密钥的一个元素）而来，该字母下面纸条对应的字母是由某个英文字母移动 j 位而来，向量 $\mathbf{A}_i$的第一项与向量 $\mathbf{A}_j$的第一项相乘得到两个字母都是 A 的可能性最大，这是因为向量 $\mathbf{A}_i$的第一项表示了任意字母移动 i 位最可能产生密文字母 A，$\mathbf{A}_j$含义相同。两个字母都是 B 的最大期望值可由向量第二项的点积求出，因此两个字母都相同的所有期望值可从 $\mathbf{A}_i \cdot \mathbf{A}_j$求出，当 $i \neq j$ 时，其值非常接近于 0.038，但当 $i=j$ 时，这个点积的结果就是 0.066。

当字母与其下方的字母通过移动达到相同时 $i=j$，也就是说，当上面的纸条移动某个值即是密钥的长度（或是密钥长度的倍数）时 $i=j$，因此在这种情况下，与正确的结果最一致。

前述密文的移位数是 5，通过比较 326 个字母发现有 24 对相同，根据刚才的分析，得出相一致的期望值是 $326 \times 0.066 \approx 21.5$，它非常接近于实际值。

2.3.3 发现密钥：第二种方法

利用前面所讲的思想，这里给出另外一种确定密钥的方法。对短一些的文本，它比第一种方法效果更好，然而它需要更多的计算量。

仍然用前面的例子来说明，为了找到密钥的长度，计算第 1、第 6、第 11 个……字母组出现的频率，同样，将它们组成向量：

$$\mathbf{V} = (0,0,7,1,1,2,9,0,1,8,8,0,0,3,0,4,5,2,0,3,6,5,1,0,1,0)$$

（第一项是字母 A 的出现次数，第二项是字母 B 的出现次数，以此类推）。如果将向量中的每一元素项除以 67，得到向量：

$$\mathbf{W} = (0,0,.1045,.0149,.0149,.0299,...,.0149,0)$$

它非常类似于 $\mathbf{A}_i$中的某个向量，这里 i 是由密钥中第一项移位而来，如果在 $0 \leqslant i \leqslant 25$ 的范围内计算 $\mathbf{W} \cdot \mathbf{A}_i$，其最大值应该来自于正确的 i 值。下面是点积的结果：

.0250，.0391，.0713，.0388，.0275，.0380，.0512，.0301，.0325，
.0430，.0338，.0299，.0343，.0446，.0356，.0402，.0434，.0502，
.0392，.0296，.0326，.0392，.0366，.0316，.0488，.0349

最大值是第三项即 .0713，它等于 $\mathbf{W} \cdot \mathbf{A}_2$，于是猜测最初的移位是 2，它对应的密钥字母是 c。

使用同样的方法去发现密钥的第 3 个元素项，使用前面列表的第 3、8、13……字母组，我们可重新计算向量 W，得出：

$$\mathbf{W} = (0,.0152,0,.0454,.0454,.0152,...,0,.0152)。$$

对于 $0 \leqslant i \leqslant 25$，$\mathbf{W} \cdot \mathbf{A}_i$ 的点积是：

.0372,.0267,.0395,.0624,.04741,.0279,.0319,.0504,.0378,
.0351,.0367,.0395,.0264,.0415,.0427,.0362,.0322,.0457,

$$.0526, .0397, .0322, .0299, .0364, .0372, .0352, .0406$$

这些数中最大的值是第四项即 .0624，它等于 $\mathbf{W}\cdot\mathbf{A}_3$，因此最可能的猜测就是最初的移位是 3，它对应的密钥字母是 d。其他的密钥元素可以采用类似的方法找到，于是得出密钥是 c, o, d, e, s。

从两个事例中我们注意到，最大的点积是相当重要的，因此我们不得不用多种猜测方法去发现正确的一个。顺便说一下，现在的方法比前述的第一种方法更高级，但第一种方法更容易实现。

为什么这种方法比第一种方法更精确呢？为了获得最大的点积，$\mathbf{W}$ 中几个大的值需要与 $\mathbf{A}_i$ 中相对大的值相匹配，在第一种方法中，我们尽量匹配的仅是 e，然后看这个选择是否对其他的字母也合适，而本方法是将所有这些一步完成。

2.4　替换密码

最流行的密码体制之一是替换密码（substitution），比如它经常使用在周末报纸的难题版块中。它的原理很简单：对字母表中的每一个字母用其他的（也可能是相同的）字母代替，更准确地说，就是用置换字母表来代替明文中的字母。在难题页面中，单词之间的空格通常保留，这最大的优点就是便于解读，因为单词的结构信息非常有用，但另一方面，为了增加安全性最好忽略空格。

移位和仿射密码是替换密码的特例，而 Vigenère 和希尔（Hill）密码不属于替换密码，因为它们一次改变的是一组字母，而不是一个字母。

大家都“知道”替换密码通过频率计算是可以被破解的，但是这个过程比人们想象的要复杂得多。

考虑下面的例子，托马斯·杰弗逊（Thomas Jefferson）有一个关于叛徒的信息要发送给本·福兰克林（Ben Franklin）。很显然，即使中途英国人截取了信息，他也不希望他们能读懂信息，于是他使用替换密码来加密信息。幸运的是，本·福兰克林知道所用的置换表，这样他就轻而易举地通过反变换获得了原始信息（当然，福兰克林非常聪明，即使先前不知道密钥他也能解密信息）。

现有假设我们是在 1776 年为英格兰下属的政府密钥及加密学校工作，需要解密下列截获信息：

```
LWNSOZBNWVWBAYBNVBSQWVWOHWDIZWRBBNPBPOOUWRPAWXAW
PBWZWMYPOBNPBBNWJPAWWRZSLWZQJBNWIAXAWPBSALIBNXWA
BPIRYRPOIWRPQOWAIENBVBNPBPUSREBNWVWPAWOIHWOIQWAB
JPRZBNWFYAVYIBSHNPFFIRWVVBNPBBSVWXYAWBNWVWAIENBV
ESDWARUWRBVPAWIRVBIBYBWZPUSREUWRZWAIDIREBNWIATYV
BFSLWAVHASUBNWXSRVWRBSHBNWESDWARWZBNPBLNWRWDWAPR
JHSAUSHESDWARUWRBQWXSUWVZWVBAYXBIDWSHBNWVWWRZVIB
IVBNWAIENBSHBNWFWSFOWBSPOBWASABSPQSOIVNIBPRZBSIR
VBIBYBWRWLESDWARUWRBOPJ1REIBVHSYRZPBISRSRVYXNFAI
RXIFOWVPRZSAEPRIKIREIBVFSLWAVIRVYXNHSAUPVBSVWWUU
```

SVBOICWOJBSWHHWXBBNWIAVPHWBJPRZNPFFIRWVV

它们的频数统计列在下表中（文本中有 520 个字母）：

W	B	R	S	I	V	A	P	N	O	...
76	64	39	36	36	35	34	32	30	16	...

英文字母近似的频率表在 2.3 节中已经给出过，下面再将有关字母的频率值列于表 2.2 中。

表 2.2　最常用英文字母的频率表

e	t	a	o	i	n	s	h	r
.127	.091	.082	.075	.070	.067	.063	.061	.060

很显然能够确定 *W* 代表 *e*（虽然 *B* 也有可能性），但其他的字母会是什么呢？我们能够猜测 *B*，*R*，*S*，*I*，*V*，*A*，*P*，*N*，可能是 *t*，*a*，*o*，*i*，*n*，*s*，*h*，*r* 中的某一两个字母，但仅凭简单的频数统计是不足以确定哪个是哪个的，现在需要做的是查看双字母组合或一对相同字母，整理这些字母，结果列在表 2.3 中（在这儿仅列出使用最频繁的字母，当然最好能列出所有的字母）。

表 2.3　双连字字母频率统计

	W	B	R	S	I	V	A	P	N
W	3	4	12	2	4	10	14	3	1
B	4	4	0	11	5	5	2	4	20
R	5	5	0	1	1	5	0	3	0
S	1	0	5	0	1	3	5	2	0
I	1	8	10	1	0	2	3	0	0
V	8	10	0	0	2	2	0	3	1
A	7	3	4	2	5	4	0	1	0
P	0	8	6	0	1	1	4	0	0
N	14	3	0	1	1	1	0	7	0

在 *W* 行和 *N* 列中的项 1 意味着 *WN* 的组合在文本中出现 1 次，在 *N* 行和 *W* 列中的项 14 意味着 *NW* 的组合出现 14 次。

我们已经确定 *W* = *e*，但如果进一步查看表中低频率的字母，就能看到 *W* 与很多这些字母都相连，它们具有 *e* 的其他特性，这有助于进一步证实我们的猜测。

元音 *a*,*i*,*o* 彼此之间很少连在一起，查看 *R* 行，就能看到 *R* 很少在 *S*,*I*,*A*,*N* 的前面，但是查看 *R* 列则看出 *R* 经常在 *S*,*I*,*A* 的后面，这样我们可以猜测 *R* 不可能是 *a*,*i*,*o* 中的字母之一，*V* 和 *N* 也不太可能是这几个字母，因为这要求 *a*，*i* 或 *o* 在 *W* = *e* 的前面频率很高才行，但似乎不太可能。进一步可以看到 *a*,*i*,*o* 最大的可能性是 *S*,*I*,*P* 中的某个。

字母 *n* 具有一个特性，即在其前面的字母 80% 是元音，因为我们已经证实 *W*，*S*，*I*，*P* 是元音，这样 *R* 和 *A* 就是最可能的候选，下面将看到哪一个是正确的。

字母 *h* 常出现在 *e* 之前，而很少出现在它的后面，这告诉我们 *N* = *h*。

最常用的连字母就是 *th*，因此 *B* = *t*。

在字母的频率表中，剩下的是 *r* 和 *s*，它们很可能等于 *V* 和 *A* 或 *R* 中的某一个，因为一

对 r 是元音，一对 s 也是很匹配的，这样我们可以断定 V 一定是 s，而 r 代表 A 或 R。

rn 的组合比 *nr* 的组合要更常用一些，而 *AR* 比 *RA* 频率要高，因此可以猜想 $A = r$ 和 $R = n$。

继续分析，可以确定 $S = o$（注意 *to* 比 *ot* 要常用得多），$I = i$ 和 $P = a$ 是最可能的选择，这样我们可以相当准确地确定文本中 520 个字母中的 382 个：

L	W	N	S	O	Z	B	N	W	V	W	B	A	Y	B	N	V	B	S
	e	h	o			t	h	e	s	e	t	r		t	h	s	t	o
Q	W	V	W	O	H	W	D	I	Z	W	R	B	B	N	P	B	P	...
	e	s	e			e		i		e	n	t	t	h	a	t	a	...

在此基础上，结合语言的知识、中等频率的字母（l，d，……），根据知识内容来填充剩余的字母。例如，在第一行中较好的猜测是 $Y = u$，因为可以组成单词 *truths*，当然这需要很多的猜测工作，需要做各种各样的假设直到最终有一个明确的结果。

鉴于前而已给出了该方法的思想，我们省略剩余部分的解密过程，加上空格（不是标点符号），解密的信息内容如下（文本是独立宣言中的一段）：

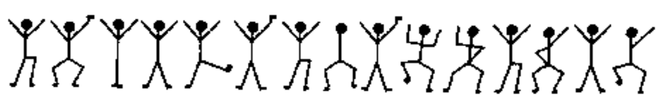
> *we hold these truths to be self evident that all men are created equal that they are endowed by their creator with certain unalienable rights that among these are life liberty and the pursuit of happiness that to secure these rights governments are instituted among men deriving their just powers from the consent of the governed that whenever any from of government becomes destructive of these ends it is the right of the people to alter or to abolish it and to institute new government laying its foundation on such principles and organizing its powers in such from as to seem most likely to effect their safety and happiness*

2.5 福尔摩斯密码

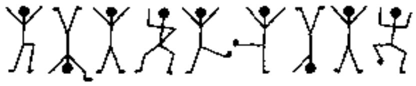
密码术在很多文学作品中都有描述，比如埃德加·艾伦·波儿（*The Gold Bug*）、威廉·撒克里（*The History of Henty Esmond*）、朱尔斯·弗恩（*Voyage to the Center of the Earth*）和阿加莎·奥斯陆（*The Four Suspects*）等的作品中均有描写。

这儿介绍一下亚瑟·柯南道尔（Arthur Conan Doyle）所写的广为流传的传奇故事，故事的主人公福尔摩斯（Sherlock Holmes）通过破解加密体制，展示了他非凡的聪明才智。在这里不过多评价这个故事，但强烈推荐读者去读其中一部分“跳舞人的冒险”（*The Adventure of Dancing Men*），下而是故事中关于密码部分的故事概要。

希尔顿·克比特（Hilton Cubitt）先生最近和埃尔希·帕特里克（Elsie Patrick）女士结婚了，他给福尔摩斯先生发去了一封信，信中有一张纸是他在 Riding Thorpe Manor 花园中发现的，这张纸是用跳舞的棒形小人所写的。



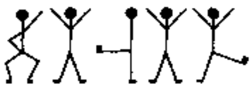
两个星期后，克比特发现有人用粉笔在他的工具间的门上写下了另外一些小人的信息：



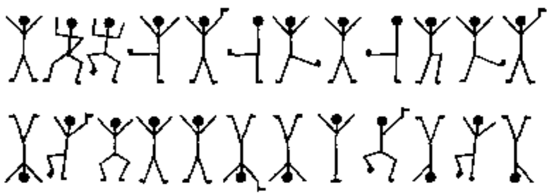
两天以后的早晨，又出现了另外的信息：



又过了三天，另一幅小人图出现了：



克比特将所有这些小人图拷贝了一份给福尔摩斯，福尔摩斯花了两天的时间进行了大量的计算，突然他从椅子上跳了起来，很显然他有了突破，于是马上发了一封长长的电报给一个人，然后就等待，并告诉沃特逊（Watson）说在随后的几天里他们很可能要去拜访克比特先生，但两天过去了却没有收到电报的回音，随后收到克比特发来的另外一封信：



福尔摩斯研究了它并且说他们马上要到 Riding Thorpe Manor 旅行一趟，没过多久给福尔摩斯的回信电报到了，福尔摩斯看后说事情已经十分危急。第二天，当福尔摩斯和沃特逊到达克比特的家时，他们发现那儿已经有了警察，克比特先生已被枪杀，他的妻子埃尔希也受了枪伤并且情况危险（后来脱险了）。福尔摩斯问了几个问题，并让人给附近爱瑞格（Elrige）农场的艾博·史兰尼（Abe Slaney）先生送去一个字条。随后福尔摩斯向沃特逊和警察解释了他是如何解密这些信息的。首先，他猜测小人图中的旗表明单词的结束，其次他注意到最普通的人是：



因此很可能是 *E*，第四个图表明-*E*-*E*-信息，很可能是 *LEVER*，*NEVER*，*SEVER* 等含义，但因为很可能这是用一个单词来回复以前的信息，福尔摩斯猜测它是 *NEVER*；其次，福尔摩斯观察下面的信息：



有形如 *E---E* 的形式，它很可能是 *ELSIE*，第 3 个信息就是---*E ELSIE*，福尔摩斯想尽了各种组合，最后得出 *COME ELSIE* 是惟一一种可能的情况。因此第一条信息是-*M*-*ERE*--*E SL*-

NE-，福尔摩斯猜测第一个字母是 *A*，第三个字母是 *H*，这就表示信息 *AM HERE A-E SLANE-*，它完整的内容应该是 *AM HERE ABE SLANEY*。第二条信息就是 *A- ELRI-ES*，当然，福尔摩斯很正确地猜出这一定表明是史兰尼所在的地方，仅剩余的字母表明的相当完整的短句就是 *AT ELRIGES*，在解密了这两条信息后，福尔摩斯发了一封电报给纽约警察局的一个朋友，这个朋友回信告诉他 Abe Slaney 是"芝加哥最危险的地方"，当最后一封信到来后，福尔摩斯解密出是 *ELSIE-RE-ARE TO MEET THY GO-*，这样他意识到空缺的字母分别是 *P*，*P*，*D*，于是福尔摩斯开始非常关注，这也就是后来他决定去 Riding Thorpe Manor 的原因。

当福尔摩斯解释完之后，警察决定立即去爱瑞格（Elrige）家逮捕史兰尼（Slaney），然而，福尔摩斯建议不必了，因为史兰尼不久就会回来，果不其然，史兰尼很快出现了并被带到了警察局。在审讯的过程时，他承认是他开的枪（他说他是自卫），并说是埃尔希·帕特里克的父亲琼（Joint）在芝加哥指使的，使用的也是他的枪，史兰尼曾经和埃尔希定了婚，但她从守卫的包围中逃了出来并到了伦敦，史兰尼最后追踪到了埃尔希所在地，并给她发了一封密信，但为什么史兰尼又掉进了福尔摩斯设的圈套呢？福尔摩斯拿出他写的字条：



从字母中已经可以推测出，信息的含义是 *COME HERE AT ONCE*，史兰尼确信这个消息是从埃尔希那儿来的，因为他可以肯定琼那儿没有人能写这样的信息，因此，他做了这次旅行，最终导致被捕。

注释

是什么使福尔摩斯在这么短的时间内解决了这个简单的替换密码问题呢？实际上对于很多这样的密码，利用频率分析和语言知识都是非常有益的，当然也需要有一点运气的，无论是在猜测的形式上还是在字母的分布上都要有一点运气。注意到绝大多数情况下 *E* 是最常用的字母，事实上在最初的四条信息中，38 个字母中 *E* 出现了 11 次，这给了福尔摩斯一个很好的开始，如果 Elsie 是 Carol，Abe Slaney 是 John Smith，解密很可能变得更困难。

鉴别是密码体制中很重要的一个问题，如果伊芙破解了艾丽斯的密码体制，那么伊芙就可能经常假扮艾丽斯和鲍勃进行通信，加强安全以阻止这种事情的发生是十分必要的，艾博·史兰尼花费了很多年去思考这个问题。

细心的读者可能已经注意到在解密信息时我们做了一点小小的欺骗，同样的符号在 *NEVER* 中代表 *V*，在 *PREPARE* 中代表 *P*，这大概是由于印刷错误，并且以后在每一个版本中复制了该错误，故事的最初版本在 1903 年发行，在最早的版本中，*NEVER* 中的 *R* 被写成了在 *ABE* 中的 *B*，但在以后的版本中得到了修正（但在后来的一些版本中，福尔摩斯所写的信息中第一个 *C* 特别弯，看起来像 *M*）。如果这些错误出现在福尔摩斯所研究的信息中，那么他解密这些信息将很困难，也不可能做出正确的推断，那么可以断定在传递中琼必须使用纠错技术。事实上，一些纠错被使用在与其相关的几乎每一个密码协议中。

2.6　Playfair 和 ADFGX 密码

在第一次世界大战中英国人和德国人分别使用 Playfair 和 ADFGX 密码，用现在的标准来看，这两个都是相当弱的密码体制，但在那个时候却需要花费相当多的努力才能攻破它们。

Playfair 体制是查尔斯·惠斯登（Charles Wheatston）先生大约在 1854 年发明的，后来他以其朋友 Baron Playfair of St. Andrews 的名字命名它，Playfair 一直致力于说服政府使用该密码。除了在第一次世界大战中使用以外，英国军队在布尔战争中也曾使用过该密码。

该密码体制的密钥是一个单词，比如 *playfair*，将单词中重复的字母去掉一个，可以得到 *playfir*，将剩下的字母排列成 5×5 矩阵的起始部分，矩阵的剩余部分则用 26 个字母表中未出现的字母来填充，*i* 和 *j* 作为一个字母来对待：

$$\begin{matrix} p & l & a & y & f \\ i & r & b & c & d \\ e & g & h & k & m \\ n & o & q & s & t \\ u & v & w & x & z \end{matrix}$$

假设明文是 *meet at the schoolhouse*。将空格忽略并将文本中每两个字母划分为一组，如果有的组中是两个相同的字母，就在中间插入一个 *x* 再重新分组，如果有必要的话在最后一组的末尾也插入 *x*。这样上述明文变成了

me et at th es ch ox ol ho us ex。

现在使用矩阵来加密每两个字母组，利用如下规则：

- 如果两个字母不在同一行或列，用该字母所在的行和另一字母所在的列对应的那个字母去代替该字母，比如，*et* 变成 *MN*，因为 *M* 和 *e* 在同一行，和 *t* 在同一列，而 *N* 和 *t* 在同一行，和 *e* 在同一列。
- 如果两个字母在同一行，用与其相邻的右边的字母代替它，对矩阵中最后一列的字母采用循环卷动，即最后一列右边的相邻的字母是该行的第一列中的字母，如 *me* 变成了 *EG*。
- 如果两个字母在同一列，用与其相邻的下面的字母代替它，矩阵卷绕从最后一行转回到第一行，如 *ol* 变成了 *VR*。

按此方法，本例的密文如下：

EG　MN　FQ　QM　KN　BK　SV　VR　GQ　XN　KU。

而要解密，就是上述的逆过程。

如果用频率攻击的方法来破解，该体制是可以被攻破的，因为在英语中各种连字（即两个字母的组合）已经被制成了表格。当然我们要看一下最常用的连字；在英语中对应的最常用的连字有：*th*，*he*，*an*，*in*，*re*，*es*，... 而且，对这些连字稍做变动又很快会产生新的结果，比如，连字 *re* 和 *er* 都很常见。如果在密文中像 *IG* 和 *GI* 这样的字母对出现得比较多的话，一个很自然的猜测就是 *e*，*i*，*r*，*g* 这些字母组成了矩阵的一个矩形形式。该密码

体制的另外一个弱点是每一个明文字母在密文中仅对应5种可能的字母，除非这个密钥很长，否则矩阵的剩余行是可以预测出来的，观察到了这些，对该体制来说仅用一段密文都可以攻破它。关于更多的密码分析可参阅［Gaines］。

ADFGX 密码过程如下，将字母表中的字母组成 5×5 的矩阵，字母 i 和 j 被认为是同一个字母，矩阵的行和列用字母 A,D,F,G,X 标记，例如，矩阵可能是：

	A	*D*	*F*	*G*	*X*
A	*p*	*g*	*c*	*e*	*n*
D	*b*	*q*	*o*	*z*	*r*
F	*s*	*l*	*a*	*f*	*t*
G	*m*	*d*	*v*	*i*	*w*
X	*k*	*u*	*y*	*x*	*h*

每一个明文字母用它所在行和列的标记代替，如 s 变成了 FA，z 变成了 DG，假设明文是

Kaiser Wilhelm。

第一步的结果就是

XA FF GG FA AG DX GX GG FD XX AG FD GA。

迄今为止可以看到，这不过是一种变相的替换密码，请注意到下一步增加了它的复杂性，选择一个关键字，比如 *Rhein*，用关键字中的字母来标记矩阵的列，将第一步的结果组成如下矩阵：

R	*H*	*E*	*I*	*N*
X	*A*	*F*	*F*	*G*
G	*F*	*A*	*A*	*G*
D	*X*	*G*	*X*	*G*
G	*F*	*D*	*X*	*X*
A	*G*	*F*	*D*	*G*
A				

现在重新调整列，使列的标记按字母表的顺序排列：

E	*H*	*I*	*N*	*R*
F	*A*	*F*	*G*	*X*
A	*F*	*A*	*G*	*G*
G	*X*	*X*	*G*	*D*
D	*F*	*X*	*X*	*G*
F	*G*	*D*	*G*	*A*
				A

最后，通过按列向下读字母（忽略标记）可得到密文如下：

FAGDFAFXFGFAXXDGGGXGXGDGAA。

只要知道了关键字解密是很容易的，从关键字的长度和密文的长度可以确定列的长度，字母被放置到了列中，重新安排顺序可以与关键字匹配，然后用初始的矩阵来恢复明文。

初始矩阵和关键字要经常改变，这样可使密码分析更困难，因为对任意一次组合仅有有

限数量的密文。然而，这个体制还是成功地被法国的密码专家乔治·潘威（Georges Painvin）和布鲁·都·切佛瑞（Bureau du Chiffre）破解，他们能够破译相当数量的信息。

这儿使用了一个技巧，假设同时截取的两份密文头几个字符是相同的，有理由猜测两份明文具有相同的单词，这意味着一份密文行和列的前几个输入与另一份相同，继续研究密文寻找相同的其他地方，这些很可能表示列的开始，如果正确的话，我们就可以知道列的长度，使用这个长度把密文划分成若干列，对第一份密文，一些列会有一个长度而另一些列会更长些，长的列就很可能靠近开始的列；其他列则靠近结束的列。对第二份密文重复上述步骤，如果一个列对两份密文都长，那它就更靠近开始列，如果列只对一份密文长而对另一份密文不长，它就是中间列，如果对两份都短，它就是结束列。按照这种方法，尝试各种列的顺序并遵照上述的限制规则，每一种顺序都与一种潜在的替换密码相对应，利用频率分析尽可能解决这些问题，最终可以得出明文和原始的加密矩阵。

之所以选择字母 *ADFGX* 是因为这些符号在莫尔斯（Morse）电码（·—，—··，··—·，——·，—··—）中是不易混淆的，从而可以避免发生传输错误，这是早期试图将纠错与密码学结合的标志之一。后来，*ADFGX* 密码被 *ADFGVX* 取代，使用 6×6 的初始矩阵，它允许使用 26 个字母加上 10 个数字。

更多关于 ADFGX 密码的体制分析，请参见［Kahn］。

2.7 分组密码

前面所介绍的密码体制中，明文中改变一个字母对应地在密文中也改变了一个字母，针对移位、仿射和替换密码，密文中给定的一个字母恰好都来自于明文中的同一个字母，这通过频率分析就非常容易发现密钥。对于 Vigenère 体制，使用的是字母的分组，对应了密钥的长度，利用频率分析要困难些，但仍然是可能的，因为每组中的各个字母没有相互作用。分组密码避免了这些问题，因为它同时加密几个字母或数字的分组。改变明文分组的一个字符，就可能改变与之相对应的密文分组潜在的所有字符。

2.6 节中所讲的 Playfair 密码是分组密码的一个简单例子，因为它是按两个字母一组加密形成两字母的分组。明文对中的一个字母改变，至少改变密文对中的一个字母，通常是改变两个字母，但是两字母的分组对安全性来说太小了，比如，利用频率分析总是可能成功破解它。

本书后面要介绍的现代密码体制的很多方法都属于分组密码的范畴，例如 DES 方法是基于 64 比特的分组，AES 使用 128 比特的分组，RSA 使用几百比特长的分组，取决于模数的使用，所有这些分组的长度都足够长，能有效地防止类似频率分析这样的攻击。

使用分组密码的标准方法就是独立地、一次性地把明文中的一块分组转换成密文的一块分组，这叫做电子电报密码本（ECB）模式，但是有一种方式是使用后续明文分组的加密分组，从密文的分组中再使用反馈机制，这就导致了密文分组的连锁（CBC）模式和密文反馈操作（CFB）模式，这些将在 4.5 节讨论。

在本书中，我们讨论希尔密码，它是由莱斯特·希尔于 1929 年发明的一种分组密码，在实践中似乎很少看见它，它的价值在于它第一次在密码学中使用了代数的方法（线性代数，模数运算），在后面的章节中将会看到，代数方法目前在我们所讨论的主题中占据了核

心地位。

选择一个整数 n，如 $n=3$，密钥是一个 $n\times n$ 的矩阵 M，它是由模 26 的整数组成的，例如，令

$$M=\begin{pmatrix}1&2&3\\4&5&6\\11&9&8\end{pmatrix}。$$

将信息表示为一系列行向量，比如，信息是 abc，我们改变它成为一个简单的行向量（0，1，2），加密就是将这个向量与矩阵相乘（一般来说，矩阵出现在乘数的右边，当然如果在左边的话，结论是类似的），并将结果模 26：

$$(0,1,2)\begin{pmatrix}1&2&3\\4&5&6\\11&9&8\end{pmatrix}\equiv(0,23,22)\pmod{26}。$$

因此，密文是 AXW（第一个字母 a 未改变是随机的，并不是这个方法的缺点）。

为了解密，我们必须确定行列式 M，使它满足

$$\gcd(\det(M),26)=1。$$

这表明，由整数元素组成的矩阵 N 要满足 $MN\equiv I\pmod{26}$，这里的 I 是 $n\times n$ 的单位矩阵。

在本例中，$\det(M)=-3$，M 的逆是

$$\frac{-1}{3}\begin{pmatrix}-14&11&-3\\34&-25&6\\-19&13&-3\end{pmatrix}。$$

因为 17 是 $-3 \bmod 26$ 的逆，我们用 17 代替 $-1/3$，并模 26 还原，得到

$$N=\begin{pmatrix}22&5&1\\6&17&24\\15&13&1\end{pmatrix}。$$

读者可以自己检验 $MN\equiv I\pmod{26}$。

关于如何计算模 n 矩阵求逆的更多内容，见 3.8 节。

要完成解密只要乘以 N 就可以了，如下：

$$(0,23,22)\begin{pmatrix}22&5&1\\6&17&24\\15&13&1\end{pmatrix}\equiv(0,1,2)\pmod{26}。$$

使用 $n\times n$ 矩阵的一般方法是，将明文分解成 n 个字符一组，并将每组转变成一个向量，该向量由 0～25 的整数组成，用 $a=0,b=1,\ldots,z=25$ 来取代字符。举个例子来说，使用前面所讲的矩阵 M，假设明文是

blockcipher。

它变换为（我们加了一个 x 以填充最后的空缺）

1 11 14 2 10 2 8 15 7 4 17 23。

现在将每一个向量乘以 M，再模 26，转换成如下的字母：

$$(1,11,14)M=(199,183,181)\equiv(17,1,25)\pmod{26}=RBZ$$

$$(2,10,2)M=(64,72,82)\equiv(12,20,4)\pmod{26}=MUE,$$

等等。

本例中，密文是

$$RBZMUEPYONOM。$$

很显然能看到，明文改变一个字母而密文总是会改变 n 个字母。比如，假设 *block* 变成了 *clock*，密文的前 3 个字母就由 *RBZ* 变成 *SDC*，这使得频率统计的有效性降低，当然如果 n 很小，不太可能让频率统计失效。两字母的组合我们叫**双连字**（diagram），三字母的组合叫**三连字**（trigram），它们的频率统计人们已经计算过了。进而，超过三字母的组合数量太多（虽然将任意普通的组合结果制成表格并不困难），而且，这样的组合出现频率又很小，以致于没有大量的文本很难得到一个有意义的数据。

既然有了密文，我们如何来解密呢？简单地将密文分成长度为 n 的组，把每组转换成向量，并右乘逆矩阵 N。本例中，我们有

$$RBZ=(17,1,25)\mapsto(17,1,25)N=(755,427,66)\equiv(1,11,14)=blo。$$

类似地，自己完成剩余的文本。

对希尔密码来说，仅仅使用密文是很难解密的，但对已知明文的攻击又很容易破解。如果不知道 n，我们可以尝试各种值直至找到正确的值为止。因此，假设 n 已知，如果我们有大小为 n 的明文的 n 个分组，这样就可以利用明文及其对应的密文得到对 M 的矩阵等式（或对 N 的，这可能更有用）。例如，假设已知 $n=2$，有明文

$$howareyoutoday=$$

$$7\quad 14\qquad 22\quad 0\qquad 17\quad 4\qquad 24\quad 14\qquad 20\quad 19\qquad 14\quad 3\qquad 0\quad 24$$

对应的密文是

$$ZWSENIUSPLJVEU=$$

$$25\quad 22\qquad 18\quad 4\qquad 13\quad 8\qquad 20\quad 18\qquad 15\quad 11\qquad 9\quad 21\qquad 4\quad 20$$

前两个分组产生的矩阵等式是

$$\begin{pmatrix}7&14\\22&0\end{pmatrix}\begin{pmatrix}a&b\\c&d\end{pmatrix}\equiv\begin{pmatrix}25&22\\18&4\end{pmatrix}\pmod{26}。$$

遗憾的是，矩阵 $\begin{pmatrix}7&14\\22&0\end{pmatrix}$ 的行列式是 -308，它模 26 是不可逆的（虽然这个矩阵可以大大减少对加密矩阵的选择数）。因此，我们替换等式的最后一行，比如用第 5 个分组，可得到

$$\begin{pmatrix}7&14\\20&19\end{pmatrix}\begin{pmatrix}a&b\\c&d\end{pmatrix}\equiv\begin{pmatrix}25&22\\15&11\end{pmatrix}\pmod{26}。$$

在这样的情况下，矩阵 $\begin{pmatrix}7&14\\20&19\end{pmatrix}$ 是模 26 的可逆矩阵：

$$\begin{pmatrix}7&14\\20&19\end{pmatrix}^{-1}\equiv\begin{pmatrix}5&10\\18&21\end{pmatrix}\pmod{26}。$$

我们得到

$$M\equiv\begin{pmatrix}5&10\\18&21\end{pmatrix}\begin{pmatrix}25&22\\15&11\end{pmatrix}\equiv\begin{pmatrix}15&12\\11&3\end{pmatrix}\pmod{26}。$$

因为希尔密码在这种情况下是很容易攻击的，所以不能认为它是非常强壮的。

针对可选择明文攻击用同样的策略进行，但它稍微快一些。而且，如果不知道 n，尝试

各种可能性最终能找到一个正确的值。于是假设 n 已知，选择的第一组明文是 $baaa\cdots$ = 1000...，第二组明文是 $abaa\cdots$ = 0100...，继续可知第 n 组明文是 ...$aaab$ = ...0001，密文的分组作为矩阵 M 的行。

针对可选择密文攻击，使用与可选择明文攻击同样的策略，在这里可选项现在代表的是密文，最终的明文将是逆矩阵 N 的行。

克劳德·香农的密码学理论基础［Shannon1］一书中有一篇基础论文，给出了一个好的密码体制能阻隔统计分析的两个特性：**扩散（diffusion）和混淆（confusion）**。

扩散意味着如果我们改变了明文中的一个字符，那么相应密文中的几个字符将要改变，类似地，如果改变密文中的一个字符，明文中的几个字符也要改变。我们看到希尔密码具有这个特性，这说明明文中字母、连字等的频率统计是漫射到密文中的几个字符的，这意味着相当多的密文需要做一个特殊意义的统计攻击。

混淆的含义是密钥并不是和密文简单地相关，特殊的情况下，每一个密文字符都依赖于密钥的几个部分，举个例子来说，假设有一个 $n \times n$ 矩阵的希尔密码，设想一下我们需要有一个长度为 n^2 的明文－密文对，才能够解出这个加密矩阵，如果改变了密文中的一个字符，矩阵的一个列将完全改变（见练习题 12）。当然，若是整个密钥改变就更理想了，如果像这样的情况发生，密码体制很可能需要同时解决整个密钥而不是一组一组分开进行。

Vigenère 和替换密码不具备扩散和混淆的特性，这也就是它们很容易用频率分析来攻击的原因。

扩散和混淆概念在任何一个设计良好的分组密码中起着非常重要的作用，当然，扩散的缺点是错误传播（这也正是密码的优点）：密文中的一个小错误在解密的信息中可以转化成一个相当大的错误，以致于通常造成解密是不可读的。

2.8 二进制数和 ASCII

在许多涉及到使用计算机的情况下，很自然地是用一串 0 和 1 来表示数据，而不是用字母和数字。

数字可以转换为二进制数（或以 2 为基数），我们很快地来复习一下，标准的记数方式是基于十进制的。举个例子来说，123 意味着 $1 \times 10^2 + 2 \times 10^1 + 3$，二进制使用 2 代替 10，仅需要数字 0 和 1，如 110101 用二进制表示 $2^5 + 2^4 + 2^2 + 1$（等于十进制数 53）。

每一个 0 或 1 叫做一个**比特（bit）**，用 8 个比特表示的数就叫做 8 比特数字，或叫一个**字节（byte）**，最大的 8 比特数字表示 255，最大的 16 比特数字能表示 65535。

通常我们涉及到的不仅仅是数字，在这种情况下单词、符号、字母和数字都要给出二进制表示，有很多可能的方式来完成这些。标准的方式之一就是 ASCII，即美国信息交换标准代码，每一个字符用 7 比特来表示，它允许标识 128 种可能的字符和符号。在计算机中通常使用 8 比特分组，基于这个原因，每个字符通常用 8 比特来表示，一旦在传输中有错误产生，8 比特数字可用奇偶校验来检测，另外也可以使用扩展字符列表，如 ü 和 è。

表 2.4 给出了 ASCII 与一些字符和标准符号的对应值，在本书中我们将使用它们，利用它们可以很方便地将文本编码成 0 和 1 的序列。

表 2.4　部分符号的 ASCII 对应码

符号	!	"	#	$	%	&	'
十进制	33	34	35	36	37	38	39
二进制	0100001	0100010	0100011	0100100	0100101	0100110	0100111
(	)	*	+	,	-	.	/
40	41	42	43	44	45	46	47
0101000	0101001	0101010	0101011	0101100	0101101	0101110	0101111
0	1	2	3	4	5	6	7
48	49	50	51	52	53	54	55
0110000	0110001	0110010	0110011	0110100	0110101	0110110	0110111
8	9	:	;	¡	=	¿	?
56	57	58	59	60	61	62	63
0111000	0111001	0111010	0111011	0111100	0111101	0111110	0111111
@	A	B	C	D	E	F	G
64	65	66	67	68	69	70	71
1000000	1000001	1000010	1000011	1000100	1000101	1000110	1000111

2.9　一次一密

一次一密（one-time pad）是一种不可攻破的密码体制，是吉伯·弗那蒙（Gilbert Vernam）和约瑟夫·摩哥那（Joseph Mauborgne）于 1918 年发明的，最初的设计是用一系列的 0 和 1 来表示信息，这可以由二进制数字来实现，比如可以利用上一节所讲的 ASCII 来表示，但这个信息应该也可以数字化成音频或视频信号。

密钥是与信息同样长度的 0 和 1 组成的随机序列，一旦密钥使用过一次就丢弃它并且不再使用，加密的过程就是信息和密钥一位一位地模 2 加，这个过程通常叫**异或（exclusive or）**，用 *XOR* 表示，换句话说，就是利用规则 0 + 0 = 0,0 + 1 = 1,1 + 1 = 0。例如，如果信息是 00101001，密钥是 10101100，可以得到如下密文：

（明文）　00101001
（密钥）　10101100
（密文）　10000101

解密使用同样的密钥，只需简单地将密文与密钥相加：10000101 + 10101100 = 00101001。

对于明文由一系列字母组成的情况，稍微有一些变化，密钥是一串随机的移位序列，每一位都是介于 0 ~ 25 之间的数字，解密使用同样的密钥，但是要减去所加上的移位。

这种加密方法对于仅知道密文的攻击是不可破解的。比如，假设密文是 *FIOWPSLQNTISJQL*，明文可能是 *wewillwinthewar*，也可能是 *theduckwantsout*，只要与明文具有相同的长度，每一种都有可能，可见从密文中得不出关于明文的信息（除了可以知道明文的长度）。关于这一点，我们讨论到关于香农熵的理论时会有更详细的描述。

如果已知一段明文，我们可以找出相应的一段密钥，但除此之外得不到关于密钥的任何其他信息。在绝大多数情况下，对可选择的明文或密文攻击都是不可能攻破的，但对于某些攻击可能暴露出密钥的一部分，通常这也是无济于事的，除非又重复使用了这部分密钥。

如何来实现这样的体制呢？它又用在什么地方呢？密钥可以事先产生，然而，产生一个正确的随机的0和1序列也是个问题。一种方法是让一些人坐在屋子里掷硬币，但对绝大部分应用来说，这太慢了；我们也可以使用盖革（Geiger）计数管，在一个小的时间间隔内数一数滴答声，如果这个数是偶数就记录0，是奇数就记录1；还有一些其他的更快的方式，但在实践中并不太满足随机性（见2.10节）；可见快速地产生一个好的密钥是相当困难的，一旦这个密钥产生了，通过可靠的方式传递到一个容器中，当需要的时候，就可以利用它们传递信息。据说，冷战时期，为了安全起见，美国和前苏联的领导人就使用了一次一密的密码体制在华盛顿特区和莫斯科之间建立了一条“热线”。

一次一密的缺点是它需要非常长的密钥，这需要花费相当大的费用去产生和传输，一旦这个密钥使用过了，如果另一条信息又重复使用它是很危险的；比如，任何关于第一条信息的知识都可能对第二条信息的破解有用。因此在大多数情况下，对于那种用很小的输入就可以产生一个相当随机的0和1序列，即称之为“近似”的一次一密。通过送信人可靠传输的信息即密钥当然比将要传递的信息在数量上小好几个数量级，有一种方法很快但不太安全，2.11节中有详细描述。

Maurer，Rabin，Ding和其他人已经开发出了各种各样的一次一密方法，设想有一个卫星能以相当快的速率产生并广播几个随机的比特序列，甚至没有一台计算机能存储哪怕是这个输出中非常小的一段，艾丽斯要传递信息给鲍勃，他们使用公开密钥方法如RSA（见第6章）共同协商从随机的比特流中取样。然后艾丽斯和鲍勃使用这些比特流作为密钥，并用一次一密方式加密信息，现在伊芙已经破解了传输使用的公钥，但由于艾丽斯和鲍勃收集的这个随机的比特流已经消失，所以伊芙不可能解密出这个信息。事实是，因为使用的是一次一密方法，她永远也不可能解密它，这样艾丽斯和鲍勃对他们所传递的信息可以获得永久的安全性。注意，对这个过程来说，有限制的存储是假设的一部分，在实际中产生和精确地取样比特流也是一个重要问题。

2.10 伪随机序列生成

一次一密和许多其他的密码学应用要求产生随机的比特流序列。在使用密码学算法如DES（见第4章）或AES（见第5章）之前，都要求产生一个随机的比特流来作为密钥。

产生随机比特流的一种方法是使用自然的随机生成，例如，半导体电阻器的热噪声是众所周知的好的随机源，但是正如抛掷硬币产生随机比特流的方法一样，由于它们在取样过程中固有的缓慢性以及确保敌人不发现这个过程的困难性，使得它们都不可行，因此我们需要有一种既能产生随机数又能由软件来实现的方法。大多数计算机都有一种用户愿意接受的产生随机数的方法，比如，标准C库函数中包括一个函数 *rand( )* ，它可产生介于0~65535之间的任意一个伪随机数，这个伪随机函数拿一粒“种子”（seed）作为输入，随后产生一个比特流的输出。

函数 *rand( )* 和许多其他的伪随机序列发生器都是建立在线性同余生成器的基础之上的，一个**线性同余生成器（Linear congruential generator）**产生一系列数 x_1，x_2，…，有关系

$$x_n = ax_{n-1} + b \pmod m。$$

x_0 是初始值，a，b 和 m 是关系式中的参数。建立在线性同余生成器基础之上的伪随机序列生成器适合于以实验为目的的情况，并不能满足密码学的要求，这是因为它们是可预知的（即使参数 a，b 和 m 未知），在这种情况下，攻击者非常可能利用某些比特知识来预测后面的比特流，事实上，已经知道的任何多项式同余生成器都存在着潜在的不安全性。

在密码学的应用中，需要一个不可预知的比特流作为输入源，现在我们要讨论两种方法来创建这样的不可预知的比特流。

第一种方法使用单向函数，这些函数 $f(x)$ 是很容易计算的，除了那些已知 y 而 $y=f(x)$ 不可求解 x 的函数。假设有这样一个单向函数 f 和一个随机的输入 s，定义 $x_j = f(s+j), j = 1,2,3,\ldots$，如果令 b_j 是 x_j 的最低有效比特，那么序列 b_0，b_1，...，将是一个伪随机的比特序列，这种随机序列生成器的方法经常使用，并且已经证明是非常可行的。利用单向函数的两个流行的加密算法是 DES（见第 4 章）和安全散列算法（见 8.3 节）。在开放 SSL 工具箱（用于 Internet 上安全通信）中有基于 SHA 的加密伪随机序列生成器的一个事例。

另外一种产生随机序列的方法是使用数论中的难题（intractable problem）。最常用的密码安全伪随机序列生成器之一是 **Blum-Blum-Shub（BBS）伪随机序列生成器**，也就是人们所知的二次方程式残数生成器，在这个规程中，首先产生两个大的素数 p 和 q，它们都是同余于 3 模 4 的，设 $n = pq$，并选择一个随机的整数 x，x 是与 n 互素的，为了初始化 BBS 生成器，设初始输入是 $x_0 \equiv x^2 \pmod n$，BBS 通过如下过程产生一个随机的序列 $b_1, b_2, \ldots$：

1. $x_j \equiv x_{j-1}^2 \pmod n$
2. b_j 是 x_j 的最低有效比特。

举例：

设：$p = 24672462467892469787$ 和 $q = 396736894567834589803$，

$n = 9788476140853110794168855217413715781961$。

令 $x = 873245647888478349013$。初始输入就是

$$x_0 \equiv x^2 \pmod n$$
$$\equiv 8845298710478780097089917746010122863172$$

$x_1, x_2, \cdots x_8$ 的值是

$$x_1 \equiv 7118894281131329522745962455498123822408$$
$$x_2 \equiv 3145174608888893164151380152060704518227$$
$$x_3 \equiv 4898007782307156233272233185574899430355$$
$$x_4 \equiv 3935457818935112922347093546189672310389$$
$$x_5 \equiv 675099511510097048901761303198740246040$$
$$x_6 \equiv 4289914828771740133546190658266515171326$$
$$x_7 \equiv 4431066711454378260890386385593817521668$$
$$x_8 \equiv 7336876124195046397414235333675005372436。$$

取上述任意一个比特串，很容易判断出这个数是奇数还是偶数，产生的序列是 $b_1, \ldots, b_8 = 0,1,1,1,0,0,0,0$。■

Blum-Blum-Shub 生成器很可能是不可预测的，详见［Stinson］。BBS 的问题是它的计算速度慢，一种提高计算速度的方法是减少 k 个 x_j 中的最低有效比特，只要 $k \leq \log_2\log_2 n$，这似乎能保证加密的安全性。

2.11 线性反馈移位寄存序列

注：本节中，所有的同余都是模 2 的。

在许多涉及加密的情况下，经常要考虑到速度和安全两者之间的平衡。如果需要高级别的安全性，往往是以牺牲速度为代价的。举个例子来说，在电缆电视中，要传输大量比特的数据，因此加密速度是非常重要的。另一方面，安全性并不总是很重要，因为为了保证系统能承受代价很高的攻击，在经济上往往是不划算的。

在本节中我们介绍一种方法，主要是在速度比安全性更重要的情况下使用的。

有一序列

$$0100001001011001111100011011101010000100101100111 1$$

可以用如下形式的初始值表述，

$$x_1 \equiv 0,\ x_2 \equiv 1, x_3 \equiv 0, x_4 \equiv 0, x_5 \equiv 0$$

和线性递归关系

$$x_{n+5} \equiv x_n + x_{n+2} \pmod 2)。$$

上述序列就是重复 31 次后的结果。

更一般的情况，考虑一个长度为 m 的线性递归关系：

$$x_{n+m} \equiv c_0 x_n + c_1 x_{n+1} + \cdots + c_{m-1} x_{n+m-1} \pmod 2,$$

这里系数 c_0，c_1，... 是整数，如果给出了**初始值（initial value）**

$$x_1, x_2, \ldots, x_m,$$

那么 x_n 的所有序列值都可以使用递归计算出来，这个由 0 和 1 组成的结果序列可以用来作为加密的密钥，也就是说，用 0 和 1 的序列来标识明文，然后将与明文位数一致的密钥比特序列与明文逐比特地进行模 2 加法运算。例如，如果明文是 1011001110001111，密钥就是前面给出的序列，我们有

（明文）	1011001110001111
（密钥）+	0100001001011001
（密文）	1111000111010110

解密只要按同样的方式给密文加上这个密钥序列就可以完成了。

这种方法的一个优点就是，一个周期非常长的密钥可以用非常少的信息来产生，这个长周期使 Vigenère 方法得到改进，就是允许用一个短周期来发现密钥。在上面的例子中，用 {0,1,0,0,0} 标识初始向量，用系数 {1,0,1,0,0} 可以产生一个周期为 31 的序列，可见用 10 比特能产生 31 比特，下面的递归展示了这个过程

$$x_{n+31} \equiv x_n + x_{n+3}$$

并且任何非零的初始向量将产生一个周期为 $2^{31}-1=2147483647$ 的序列。因此，62 比特可以产生周期大于 2Gbit 的密钥，比起一次一密来说这是一个非常大的优点，当然，在这儿全部的 2Gbit 事先应被传输过去。

使用人们所知的硬件**线性反馈移位寄存器（LFSR）**，该方法在实践中是很容易实现的，并且速度很快。在图 2.1 中，我们描述了一个简单的线性反馈移位寄存器，对于更复杂的递

归，要实现需使用更多的寄存器和异或（XOR）运算器。

对每增加一次计算，每一个盒子中的比特被移到另一个盒子中作为输入，用⊕表明其引入的比特位的模 2 加法。输出即比特 x_m 和明文的下一个比特相加以产生密文，图 2.1 表示了 $x_{m+3} \equiv x_{m+1} + x_m$ 的递归，一旦 x_1, x_2, x_3 的原始值给定后，这个机器就可以高效率地产生比特流。

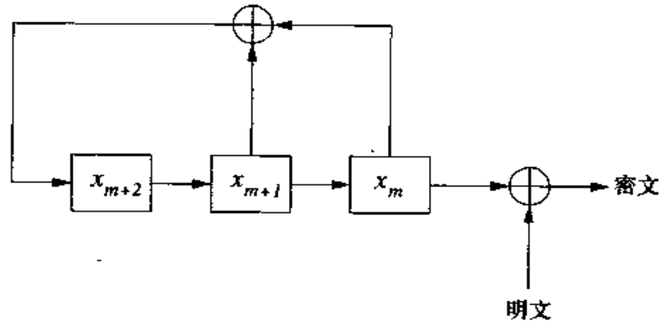


图 2.1 一个满足 $x_{m+3} = x_{m+1} + x_m$ 的线性反馈移位寄存器

不幸的是，前述的加密方法在已知明文的攻击中是很容易被攻破的，更进一步地说，只要我们知道了一段连续的明文比特以及其相对应的密文比特，就可以确定这个递归关系，因此也就可计算所有关于密钥的比特序列。通过减（或加；它模 2 的结果是相同的）这个明文，明文是从密文中模 2 得到的，就得到了密钥的比特串。因此，剩下的讨论我们将忽略明文和密文，并假定已经发现了密钥序列的一部分。

例如，假设我们知道具有周期为 15 的序列 0110101111000100110101111... 初始的一段 011010111100，并且假定知道序列是由线性递归产生的，那么怎样来确定这个递归的系数呢？我们甚至不需要知道长度，这样需从长度 2 开始（长度 1 会产生一个常数序列），设想这个递归是 $x_{n+2} = c_0 x_n + c_1 x_{n+1}$，令 $n = 1$ 和 $n = 2$，并使用已经知道的值 $x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 0$，得到等式

$$1 \equiv c_0 \cdot 0 + c_1 \cdot 1 \ (n = 1)$$

$$0 \equiv c_0 \cdot 1 + c_1 \cdot 1 \ (n = 2)$$

表示成矩阵的形式，就是

$$\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \end{pmatrix} \equiv \begin{pmatrix} 1 \\ 0 \end{pmatrix}。$$

能解出 $c_0 = 1, c_1 = 1$，这样我们就可以猜出递归是 $x_{n+2} \equiv x_n + x_{n+1}$，然而这并不正确，因为 $x_6 \not\equiv x_4 + x_5$。因此，我们再尝试长度 3，结果矩阵等式是

$$\begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \end{pmatrix} \equiv \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}。$$

矩阵的行列式是 0 模 2；实际上这个等式无解。可以看出，这是因为矩阵中每一列的和是 0 模 2 的，这样右边的向量就不存在。

现在考虑长度 4，矩阵等式如下：

$$\begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} \equiv \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix}。$$

解出 $c_0 = 1, c_1 = 1, c_2 = 0, c_3 = 0$。递归的结果现在推测出是

$$x_{n+4} \equiv x_n + x_{n+1}$$

用这个结果产生的剩下的密钥段在前面是已知的，因此这正是产生这个密钥的最好的猜测。事实上，上述的快速计算已经表明了这个事实，这样就发现了递归关系。

更一般的表示如下，为了测试长度为 m 的一个递归，假定我们知道 $x_1, x_2, \ldots, x_{2m}$，矩阵等式是

$$\begin{pmatrix} x_1 & x_2 & \cdots & x_m \\ x_2 & x_3 & \cdots & x_{m+1} \\ \vdots & \vdots & \ddots & \vdots \\ x_m & x_{m+1} & \cdots & x_{2m-1} \end{pmatrix}\begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{m-1} \end{pmatrix} \equiv \begin{pmatrix} x_{m+1} \\ x_{m+2} \\ \vdots \\ x_{2m} \end{pmatrix}。$$

后面将会看到，矩阵是模 2 可逆的，当且仅当没有一个长度小于 m 的线性递归式满足 $x_1, x_2, \ldots, x_{2m-1}$。

寻找递归式的系数的策略现在已经清楚了，假设我们知道密钥的前 100 比特，对 $m = 2, 3, 4, \ldots$，形成前面所讲的 $m \times m$ 矩阵，并且计算它的行列式。如果 m 中有几个连续的值产生的行列式是 0，就停止计算，最后产生非零（即 1 模 2）行列式的值很可能就是递归式的长度，求解这个矩阵等式，可能得出系数 $c_0, \ldots, c_{m-1}$。然后就可以检查这个递归式产生的序列是否与所知道的密钥的比特串匹配，如果不匹配，再试大点的 m 值。

假设我们不知道开始的 100 比特，但知道密钥中另外一些连续的 100 比特，和上面有同样的应用过程，使用这些比特作为开始位置。实际上，一旦找到了这个递归式，就可以回溯寻找开始位置之前的比特串。

这儿有一个例子，假设有下列 100 比特的序列：

1001100100111000110001010001111011001111101010100l

0110110101100001101110010101111000000010001001000。

前 20 个的行列式从 $m = 1$ 开始，是

1,0,1,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0。

合理的猜测是 $m = 8$ 给出了最后的非零行列式，解出关于系数的矩阵等式，可以得到

$$\{c_0, c_1, \ldots, c_7\} = \{1,0,1,0,1,1,1,1\},$$

这样可以猜出递归式是

$$x_{n+8} \equiv x_n + x_{n+2} + x_{n+4} + x_{n+5} + x_{n+6} + x_{n+7}。$$

这个递归式生成了原始序列的所有 100 项，都有正确的值，至少基于已有的知识它是正确的。

假设这个 100 比特串在某一序列的中间，而我们想知道前面的比特串。例如，假设序列从 x_{17} 开始，于是有 $x_{17}=1$，$x_{18}=0$，$x_{19}=0$，…，写出递归式如下：

$$x_n \equiv x_{n+2} + x_{n+4} + x_{n+5} + x_{n+6} + x_{n+7} + x_{n+8}$$

（也可能会有一些符号错误，但想到我们是工作在模 2 的方式下，因此，$-x_n \equiv x_n$，$-x_{n+8} \equiv x_{n+8}$）。令 $n = 16$，得到

$$\begin{aligned} x_{16} &\equiv x_{18} + x_{20} + x_{21} + x_{22} + x_{23} + x_{24} \\ &\equiv 0 + 0 + 0 + 1 + 0 + 1 \equiv 0。\end{aligned}$$

继续这种方式，可以成功确定 $x_{15}, x_{14}, \ldots, x_1$。

下面来证明这个结果。

命题：令

$$M = \begin{pmatrix} x_1 & x_2 & \cdots & x_m \\ x_2 & x_3 & \cdots & x_{m+1} \\ \vdots & \vdots & \ddots & \vdots \\ x_m & x_{m+1} & \cdots & x_{2m-1} \end{pmatrix},$$

如果序列 $x_1, x_2, \ldots, x_{2m-1}$ 满足一个长度小于 m 的线性递归式，那么 $\det M \equiv 0$，反过来，如果序列 $x_1, x_2, \ldots, x_{2m-1}$ 满足长度为 m 的线性递归式，并且 $\det M \equiv 0$，那么序列也满足长度小于 m 的一个线性递归式。

证明：

首先我们在递归式的长度上做一些说明，以便解释命题中的最后一句话。一个序列可能满足一个长度为 3 的关系，如 $x_{m+3} \equiv x_{m+2}$，它同时也可能满足短一点的关系，如 $x_{m+1} = x_m$（至少对 $m \geqslant 2$），但并不能明显地看出，一个序列可能满足长度小于某个期望值的递归式。例如，考虑关系 $x_{n+4} \equiv x_{n+3} + x_{n+1} + x_n$，假设最初的序列值是 1，1，0，1。通过这个递归式计算出后来的项是：1,0,1,1,0,1,1,0,1,1,0,1...。很容易看到这个序列满足 $x_{n+2} \equiv x_{n+1} + x_n$。

如果有一个长度小于 m 的递归式，那么矩阵的一行就是其他行的线性组合。比如，如果递归多项式是 $x_{n+3} = x_{n+2} + x_n$，那么，第 4 行是第 1～3 行之和，因此，行列式是 0 模 2 的。

相反，假设行列式是 0 模 2 的，那么存在一个非零的行向量 $\bar{b} = (b_0, \ldots, b_{m-1})$ 满足 $\bar{b}M \equiv 0$，这就给出了一个递归的关系，但并不能立即看出，对所有 x_{2m-1} 均满足这种递归关系。例如，假设矩阵 M 的最上面两行之和等于第三行，对 $n = 1, 2, \ldots, m$ 有 $x_{n+2} \equiv x_n + x_{n+1}$，我们需要将这种关系扩充至，对 $n = 2m-3$ 有 $x_{2m-1} \equiv x_{2m-3} + x_{2m-2}$，刚才我们已假定，有长度为 m 的递归式，对 $0 \leqslant n < m$，有 $x_{n+m} = c_0 x_{n+1} + \cdots c_{m-1} x_{n+m-1}$。通过定义 $x_{2m}, x_{2m+1}, x_{2m+2}, \ldots$ 来扩充这个等式（当然，如果这个序列已经有 x_n 项，定义 $n \geqslant 2m$，这可能与我们证明使用的临时变量是不同的）。这样可以得到矩阵等式

$$M_n \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{m-1} \end{pmatrix} \equiv \begin{pmatrix} x_{n+1} & x_{n+2} & \cdots & x_{n+m} \\ x_{n+2} & x_{n+3} & \cdots & x_{n+m+1} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n+m} & x_{n+m+1} & \cdots & x_{n+2m-1} \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{m-1} \end{pmatrix} \equiv \begin{pmatrix} x_{n+m+1} \\ x_{n+m+2} \\ \vdots \\ x_{n+2m} \end{pmatrix}$$

（这里 M_n 表示中间表达式的 $m \times m$ 矩阵），注意，M_0 是矩阵 M 的初始矩阵，对大一些的 n 值，它仅表达这个递归关系，回想刚才的向量 $\bar{b}$ 满足 $\bar{b}M \equiv 0$。令前面等式中的 $n = 0$，并且在左边乘以向量 $\bar{b}$，于是 $\bar{b}M_0 \equiv \bar{b}M \equiv 0$，左边相乘得出零，右边也是一样，这就意味着 $\bar{b} \cdot (x_{m+1}, \ldots, x_{2m}) \equiv 0$。

现在考虑 $n=1$ 的情况，刚才已经证明 $\bar{b}$ 乘 M_1 的最后一列是 0，而 M_1 的其他列是从 M_0 列得来的，因此 $\bar{b}$ 也同样与其他的列相乘为 0，因此，$\bar{b}M_1 \equiv 0$。刚才的证明似乎隐含着 $\bar{b}$ 乘 M_2 的最后一列也是 0，以这种方式继续，可以看到对所有 n 都有 $\bar{b}M_n \equiv 0$。

很容易看到，这样就产生了一个长度小于 m 的递归式。例如，如果 $\bar{b}=(1,1,1,0,0,\ldots)$，那么对所有 n，有 $x_{n+2}\equiv x_n+x_{n+1}$，特别地，对 $n\leqslant 2m-3$ 也成立（人们可能认为这个关系式应该是 $x_{n+2}+x_n+x_{n+1}\equiv 0$，但前面讲过是针对模 2 的，所以 + 和 - 是相同的）。而且，注意到长度为 m 的一个向量给出了长度至多为 $m-1$ 的一个递归式（因为最大的非零项继续在等式的另一边）。这就完成了证明。 □

最后，再介绍一下关于序列串的周期。设想递归式的长度是 m，任何序列串中 m 个连续项可以确定后而的值，反过来往前推算递归式，也可以计算前面的值。很明显，如果有 m 个连续的 0，那么所有后面的值是 0，所有前面的值也是 0，因此，我们排除这种情况。有长度为 m 的由 0 和 1 组成的 2^m-1 个序列串，其中至少有一项非零。因此，只要有多于 2^m-1 项，一定会生成 2 倍的长度为 m 的序列串，可见序列是重复的，序列的周期至多是 2^m-1。

结合一个递归多项式 $x_{n+m}\equiv c_0x_n+c_1x_{n+1}+\cdots+c_{m-1}x_{n+m-1} \pmod 2$，有多项式

$$f(T)=T^m-c_{m-1}T^{m-1}-\cdots-c_0,$$

如果 $f(T)$ 是模 2 不可约的（这意味着它不能分解为两个低次多项式），就表明这个周期划分为 2^m-1。有趣的是当 2^m-1 为素数时（这些叫做 Mersenne 素数）。如果周期不是 1，也就是说如果这个序列不是常数，那么可以确定在这种情况下，周期一定是最大的，即 2^m-1，周期为 $2^{31}-1$ 就属于这种情况。

要更深入地学习线性反馈移位寄存序列，可参见 [Golomb] 或 [van der Lübbe]。

阻止上述攻击的方式之一就是使用非线性递归多项式，例如

$$x_{n+3}\equiv x_{n+2}x_n+x_{n+1}。$$

一般来说，这些体制比较难攻破，但在这里我们不去讨论它。

2.12 Enigma

我们所知的转轮机加密设备是在 1920 年发明的，最著名的设计就是由亚瑟·谢尔必斯（Arthur Scherbius）发明的 Enigma，它是二战期间德国使用的最出名的机器之一。

据说它非常安全，许多人企图攻破它，最终都失败了。然而，三位波兰密码学家麦仑·瑞杰韦斯克（Marian Rejewski）、海瑞克·盖尔斯克（Henryk Zygalski）和杰瑞·洛椰克（Jerzy Różycki）在 1930 年成功地破译了 Enigma 的早期版本，1939 年他们所使用的方法传到了英国，正好早于德国侵略波兰两个月，英国发展了波兰人的这些技巧，并在整个第二次世界大战期间成功地破译了德国的信息。

战争结束后几乎经过了 30 年，Enigma 已被破译仍然是个不为人们所知的秘密，部分原因是由于英国收买和逮捕了 Enigma 的发明者，并且不让他们知道该密码体制已被攻破。

接下来我们简要描述一下 Enigma，并介绍瑞杰韦斯克所发明的破译方法。如果想更进一步查看实例，可参阅［Kozaczuk］，这本书有瑞杰韦斯克对 Enigma 攻击的详细描述。

这个机器的示意图在图 2.2 中给出。

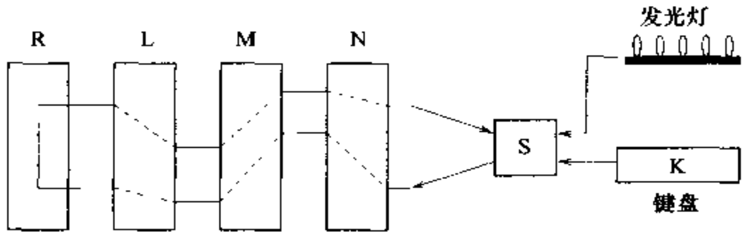


图 2.2　Enigma 机器的示意图

图中 L，M，N 为转动子。每个转动子的一边是 26 个固定的电子触点，这些触点圈形排列，转动子的另一边是 26 个载荷簧触点，也是按圈形排列，以便子和邻近的固定触点相连。在每一个转动体的内部，固定触点与载荷簧触点可随意相连，每一个转动子的内部连接都是不同的，每一个都有 26 种可能的初始设置。

R 是回动鼓，它的 26 个载荷簧触点成对相连。

K 是键盘，和打字机键盘相同。

S 是线路连接板，它有大约 6 对插头可用来和 6 组字母相互交换。

当按下一个键时，第一个转动子 N 转动 1/26 圈，然后，从这个键开始，电流通过 S，再传送到 N，M，L 上，当电流到达回动鼓 R 时，电流又通过另一路径返回到 L，M，N，后到达 S。这时，电流点燃了与键盘上一个字母相对应的球状体，这就是加密的字母。

因为转动子 N 是在每次加密之前开始转动，这比替换密码要复杂得多，而且转动子 L 和 M 也要转动，这就更少见了，就像汽车中的里程表一样。

解密使用完全相同的方法。假设发送方和接收方使用相同的机器，两个都设置成同样的初始位置，发送方通过在键盘上键入加密信息，并利用灯来记录字母的顺序，这个密文随后传送给了接收方，它也将密文键入到机器上，在灯上所反映的字母顺序就是原始的信息。下面可以看到这一点，灯"a"和键"a"在线路连接板上是通过一条金属线连接的，灯"h"和键"h"在线路连接板上也是通过一条金属线连接的，如果按了键"a"而灯"h"亮了，那么通过机器的电流路径就将灯"a"与键"h"相连。因此，如果按的是"h"键，那么"a"键的灯就会亮。

类似的原因表明，没有一个字母曾经加密成它自己，这看起来是个不错的主意，但实际上也有一些不足，因为对密码分析家来说，一开始他们就可以排除掉很多种可能的情况。

该体制的安全性主要依赖于对转动子最初设置的保密程度，即线路连接板上插头的设置、转动子和回动鼓的内部配线设置等。转动子和线路板的设置需要周期性地改变（比如每天变动一次）。

然而，只要给出足够的密文，仍然存在推测信息的方法，实际中已经有人这么做过。

有多少种这样的设置呢？3 个转动子每一个都有 26 种初始值，有 $26^3=17576$ 种可能性，3 个转动子有 6 种可能的排列，那么转动子就有 $6\times17576=105456$ 种可能的初始值。在后来的 Enigma 版本中，有 5 个可利用的转动子，每天选择 3 个，这就有 60 种可能的转动子序

列，于是有 1054560 种转动子方式。

在线路连接板上，有 100391791500 种交换 6 对字母的方式。

总体来看，这个机器有太多种可能的初始方式，似乎没有攻破的希望，采用频率分析的方法会以失败告终，因为转动子的旋转对信息中的每一个字符都改变了它们的置换结果。

这样一来，如何能够攻破 Enigma 呢？在这儿我们不给出整个破译过程，但还是要介绍一下在 1937 年左右发现的转动子的初始设置是如何被确定的，这种攻击的方法依赖于在那个时期所使用的协议的弱点，但它给出了在其他情况下如何攻击的一般方法。

每个 Enigma 操作员都有一本密码书，它主要包括下个月每天要使用的设置，但是，假如这些设置不加修改地使用，那么在某一天发送的每一条信息就可能存在它的第一个字母是用同样的替换密码加密的情况，转动子然后转动，每一个文本的第二个字母可能对应另一个替换密码，这种替换对那一天的所有信息可能都是相同的，将一天内截取的每一条信息的第一个字母进行频率分析，就很可能破译出每条信息的第一个字母，第二次频率分析可能破译第二个字母，类似地，分析密文中剩余的字母（除了最长的密文中的结尾部分）就解密了信息。

为了避免这样的问题，操作员为每一条信息选择一个信息密钥，这个信息密钥由 3 个字母组成，比如，*r*，*f*，*u*，然后，他们再从密码本上使用每天的设置来加密这个信息密钥，但由于无线通信有可能会出错，所以要重复写一次 *rfu*，这样加密 *rfurfu* 以得到一个六字母的字串。随后将转动子设置在 *r*，*f* 和 *u* 的位置，就可以开始实际信息的加密了，可见，最初发送的六字母是加密密钥，剩余的是密文，由于每一条信息使用了不同的密钥，所以频率分析是不太可能的。

对接收方来说，只需要使用密码本上每天的设置来解密信息的前六个字母，然后他们重设置转动子，以与解密密钥指明的位置吻合，最后进行信息的解密处理。

密钥的副本是密码分析家要攻破的一个主要目标，设想某一天截获了一些信息，其中有下列初始六字母中的三个：

dmqvbn

vonpuy

pucfmq

所有这些都是用密码本中同一天的设置来加密的，第一个字母的加密对应于 26 个字母的置换，我们称之为置换 A，在第二个字母加密之前，转动子转动，这样第二个字母就使用另一个置换，叫置换 B。类似地，对剩下的四个字母的置换分别是 C，D，E，F，这个策略看起来是乘积 AD，BE 和 CF。

在这里需要介绍一下置换的有关问题。当将两个置换 A 和 D 写成 AD，意味着先做置换 A 再做置换 D（有的书使用相反的顺序）。对于 a 到 b，b 到 c，c 到 a 的置换，用3-环（abc）（3-cycle（abc））表示。用类似的符号来标明其他长度的递归，如（ab）表示置换 a 到 b，置换可以写成一个环的乘积。例如，如下置换：

$$(dvpfkxgzyo)(eijmunqlht)(bc)(rw)(a)(s)$$

代表置换 d 到 v，v 到 p，t 到 e，r 到 w，等等，且 a 和 s 保持不变。如果环是不相连的（意味着通常情况下没有两个环有多个字母），那么就可以分解为惟一的环。

让我们来看一下截获的文本。我们并不知道这三个信息密钥的字母组成，但让我们命名第一个信息密钥是 *xyz*。因此，*xyzxyz* 加密成了 *dmqvbn*，已经知道置换 A 发送 x 到 d，又有第

四个置换 D 发送 x 到 v。我们还知道一些信息，由于机器的内部配线，对置换 A 实际交换的是 x 和 d，而 D 交换的是 x 和 v（即，A 发送 d 到 x，而 D 发送 x 到 v）。未知的 x 就已经被排除。类似地，第二个截获的文本告诉我们 AD 发送 v 到 p，从第三个文本可知 AD 发送 p 到 f。因此可以确定

$$AD = (dvpf\cdots)\cdots$$

用同样的方法，从三条信息的第二和第五个字母可知

$$BE = (oumb\cdots)\cdots$$

从第三和第六个字母可知：

$$CF = (cqny\cdots)\cdots$$

利用这些信息，我们足以推出 AD，BE 和 CF 的乘积环的分解。例如有：

$$AD = (dvpfkxgzyo)(eijmunqlht)(bc)(rw)(a)(s)$$
$$BE = (blfqveoum)(hjpswizrn)(axt)(cgy)(d)(k)$$
$$CF = (abviktjgfcqny)(duzrehlxwpsmo)。$$

这段信息仅依赖于线路板和转动子每天的设置，并不依赖于信息密钥，因此，它只与给定的某一天所使用的每一台机器有关。

下面来看一下线路连接板的影响。先介绍一下在处理过程中开始的置换 S，然后在处理过程末尾增加逆置换 S^{-1}。关于置换的另外一个事实是：假设有一个置换 P，以及对某一个置换 S 的另一个形式为 SPS^{-1} 的置换（这儿 S^{-1} 表示置换 S 的逆；在本例中，$S=S^{-1}$），把它们分解成环，它们将总是不具有相同的环，但在分解中环的长度相同。比如，AD 有长度为 10，10，2，2，1，1 的环。如果对任意的置换 S，分解 $SADS^{-1}$ 成环。将又得到长度为 10，10，2，2，1，1 的环。因此，如果线路连接板的设置改变了，但转动子的初始位置依然相同的话，那么环的长度仍然不会改变。

可能你已经注意到了，在 AD，BE 和 CF 分解成环的式子中，环的长度几乎都是偶数，这是一般的现象，详细解释请参见前面提到的 Kozaczuk 的书中的附录 E。

瑞杰韦斯克和他的同事将转动子的所有 105456 种初始设置汇编成册，这 105456 种初始设置是由三种置换 AD，BE，CF 的环长度设置对应面来的。在这种情况下，他们使用某一天的密文，推测出环的长度，并找出一个小范围内转动子的初始设置，每一种置换都要单独测试。发现线路板的影响（当使用正确的设置时）不过是一种替换密码，它是很容易攻破的。直到 1938 年 9 月，采用了一种新修改的方式发送信息密钥，上述这种方法才不被使用。方法的改进主要是又一次解密信息，这个过程也是机械化的，使用一种叫着“炸弹”的机器来发现每天的密钥，每次需要两小时。

这些方法后来在第二次世界大战中又由 Bletchley Park 的英国人进一步改进了，包括创造更高级的“炸弹”。这些机器是艾伦·图灵（Alan Turing）设计的，人们常认为这就是第一代电子计算机。

2.13 习 题

1. 凯撒与马可·安东尼要举行一个秘密会议，要么在 Tiber（河流名），要么在 Coliseum

（剧场、竞技场名），他发送了密文 *EVIRE*，但是安东尼并不知道密钥，因此他尝试所有的可能性。试问，在哪儿能遇到凯撒？

2. 使用仿射函数 $5x+7 \pmod{26}$ 加密 *howareyou*。解密函数是什么？解释它的工作过程。

3. 考虑一个仿射密码（mod 26），使用 *hahaha* 进行一个选择明文的攻击，密文是 *NONONO*，确定这个加密函数。

4. 下列密文是通过一个仿射密码模 26 来进行加密的：

CRWWZ

明文以 *ha* 开头，请解密该信息。

5. 假设使用仿射密码加密信息，然后使用另一个仿射密码加密密文（两个均工作在模 26 下），请问这样做比只用一个仿射密码有什么优点吗？请说明为什么。

6. 假设对仿射密码用模 27 代替模 26，有多少种可能的密钥？如果是模 29 呢？

7. 请利用函数 $\alpha x+\beta$,已知 $\gcd(\alpha,26)=d>1$ 实现一个仿射加密,试证明如果 $x_1=x_2+(26/d)$,那么 $\alpha x_1+\beta\equiv\alpha x_2+\beta \pmod{26}$。并证明在这种情况下不能够得到惟一的解密。

8. 假设有一段仅有 3 个字母 *a*，*b*，*c* 组成的文字，它们出现的频率分别为 .7，.2，.1。下面的密文是通过 Vigenère 方法加密的（当然移位是模 3 而不是模 26）：

CAAABBCACBCABACAABCCCACA。

说明密钥的长度很可能是 2，并确定最可能的密钥。

9. 如果 $\mathbf{v}$ 和 $\mathbf{w}$ 是两个 n 维向量，$\mathbf{v}\cdot\mathbf{w}=|\mathbf{v}||\mathbf{w}|\cos\theta$，这里 θ 是两个向量的夹角（是通过两个向量在二维平面上测量的），$|\mathbf{v}|$表示 $\mathbf{v}$ 的长度。利用上述条件及 2.3 节中的符号说明，当 $i=0$ 时点积 $\mathbf{A}_0\cdot\mathbf{A}_i$是最大的。

10. 密文 *YIFZAM* 是通过希尔密码利用矩阵 $\begin{pmatrix}9 & 13\\2 & 3\end{pmatrix}$ 加密的，请找出明文。

11. 密文文本 *GEZXDS* 是通过希尔密码用 2×2 矩阵加密得来的，明文是 *solved*，请找出加密矩阵 M。

12. (a) 密文文本 *ELNI* 是通过希尔密码用 2×2 矩阵加密得来的，明文是 *dont*，请找出加密矩阵 M。

(b) 假设密文是 *ELNK*，明文仍然是 *dont*，请找出加密矩阵。注意矩阵的第二列改变了。这表明加密矩阵整个第二列涉及到密文的最后一个字符（见 2.7 节最后）。

13. 三元 *LFSR* 序列是以 001110 开始的，请找出这个序列后面的 4 个元素。

14. 考虑一个序列以 $k_1=1, k_2=0, k_3=1$ 开始，并由三元递归式 $k_{n+3}=k_n+k_{n+1}+k_{n+2}$ 定义而来。这个序列也可由二元递归式得出，通过建立和解适当的矩阵等式，确定这个二元递归式。

15. 在 20 世纪 80 年代中期，NSA（国家安全局）的征兵广告的最上面出现了 1 之后跟了 100 个 0。文本的开始是："你正在看一个'10 的 100 次方'，10 进行到 100 次方，后面跟 100 个 0，一天数 24 个小时，需要数 120 年才能数到 10 的 100 次方，需要两代人。这是一个不可能达到的数，这个数超出了我们的想象。"为了在 120 年内数到 10 的 100 次方，每秒钟需要数多少个数？（为达到上述目标，一种猜测是，广告公司假定达到 100 位数的时间与数到 10 的 100 次方的时间相同。）

16. 艾丽斯使用下面的密码体制之一发送信息给鲍勃，实际上令艾丽斯烦恼的是她的明文由一个字母（只有她自己知道）重复几百次组成，伊芙知道使用的密码体制，但不知道密钥，并截获了密文。对体制（a）和（b），试阐明伊芙怎样能意识到明文是由一个字母重复组成的，并确定伊芙是否能推测出这个密钥。对体制（c），假设伊芙猜出了密文是由一个字母重复组成的，说明伊芙怎样才能推测出密钥。

（a）移位密码

（b）仿射密码

（c）Vigenère 密码（假设密钥是长度为 8 ~ 12 个字母组成的一个英语单词）

2.14 上 机 题

1. 下列密文是由移位密码加密得来的：

ycvejqwvhqtdtwvwu

请解密（明文存储在文件名 *ycve* 中）。

2. 下列密文是移位密码的输出结果：

lcllewljazlnnzmvyiylhrmhza

通过执行频率统计，猜出密文所使用的密钥，使用计算机来验证你的结果，解密出的明文是什么？（密文存在文件 *lcll* 中。）

3. 下列密文是用仿射密码加密的：

edsgickxhuklzveqzvkxwkzukcvuh

明文开始的两个字母是 *if*，请解密（密文存储在文件 *edsg* 中）。

4. 下列密文是通过仿射密码，利用函数 $3x+b$，b 取某些值得来的：

tcabtiqmfheqqmrmvmtmtaq

请解密（密文存放在文件 *tcab* 中）。

5. 用仿射密码 $y\equiv mx+n \pmod{26}$，取 $m>26$ 来上机实践加密。另外，判断这些加密是否与 $m<26$ 得到的加密结果相同。

6. 通过编程完成该题。编写代码来创建一个新的字母表 $\{A,C,G,T\}$。例如，这个字母表能够反映四种核苷酸，即腺嘌呤、胞核嘧啶、鸟嘌呤和胸腺嘧啶，它们都是 DNA 和 RNA 代码的基本组成部分，数字 0，1，2，3 分别与字母 A，C，G，T 对应。

（a）使用一位移位密码加密下列关于核苷酸的序列，它来自于人类第 13 对染色体：

GAATTCGCGGCCGCAATTAACCCTCACTAAAGGGATCTCTAGAACT。

（b）写出一个程序，能在核苷酸的字母表中完成仿射密码，在仿射密码中有什么限制？

7. 下列序列是利用密钥长度至多为 6 位的 Vigenère 密码完成加密的，请解密该信息，并确定明文中有何特别之处，它是怎样影响结果的？

hdsfgvmkoowafweetcmfthskucaqbilgjofmaqlgspvatvxqbiryscpcfr
mvswrvnqlszdmgaoqsakmlupsqforvtwvdfcjzvgsoaoqsacjkbrsevbel
vbksarlscdcaarmnvrysywxqgvellcyluwwveoafgclazowafojdlhssfi
ksepsoywxafowlbfcsocylnqqsyzxgjbmlvgrggokgfgmhlmejabsjvgml
nrvqzcrggcrghgeupcyfqtydycjkhqluhgxgzovqswpdvbwsffsenbxapa
sgazmyuhgsfhmftayjxmwznrsofrsoaopgauaaarmftqsmahvqecev

（密文存储在文件 *hdsf* 中，明文来自于欧内斯特·文森特·莱特（Ernest Vincent Wright）所写的 *Gadsby*。）

8. 下列文字是用 Vigenère 方法加密而得到的，请找出明文。

```
ocwyikoooniwugpmxwktzdwgtssayjzwyemdlbnqaaavsuwdvbrflauplo
oubfgqhgcscmgzlatoedcsdeidpbhtmuovpiekifpimfnoamvlpqfxejsm
xmpgkccaykwfzpyuavtelwhrhmwkbbvgtguvtefjlodfefkvpxsgrsorvg
tajbsauhzrzalkwuowhgedefnswmrciwcpaaavogpdnfpktdbalsisurln
psjyeatcuceesohhdarkhwotikbroqrdfmzghgucebvgwcdqxgpbgqwlpb
daylooqdmuhbdqgmyweuik
```

（密文存放在文件 *ocwy* 中，明文来自于亚瑟·柯南道尔所写的“跳舞人的冒险”。）

9. 下列文字是用 Vigenère 方法加密而得到的，请找出明文（密文存储在文件 *xkju* 中）。

```
xkjurowmllpxwznpimbvbqjcnowxpcchhvvfvsllfvxhazityxohulxqoja
axelxzxmyjaqfstsrulhhucdskbxknjqidallpqslluhiaqfpbpcidsvcih
whwewthbtxrljnrsncihuvffuxvoukjljswmaqfvjwjsdyljogjxdboxaju
ltucpzmpliwmlubzxvoodybafdskxgqfadshxnxehsaruojaqfpfkndhsaa
fvulluwtaqfrupwjrszxgpfutjqiynrxnyntwmhcukjfbirzsmehhsjshyo
nddzzntzmplilrwnmwmlvuryonthuhabwnvw
```

10. 下列是使用希尔密码加密的密文：

```
zirkzwopjjoptfapuhfhadrq
```

使用的矩阵是

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 2 & 1 \\ 11 & 2 & 4 & 6 \\ 2 & 9 & 6 & 4 \end{pmatrix},$$

请解密。

11. 下面的序列是通过一个线性反馈移位寄存器产生的，确定产生它的递归式。

1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1,
0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1,
1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1

（序列存放在文件 *L101* 中。）

12. 下列是一个 LFSR 输出的前 100 项，找出这个递归式的系数。

1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0,
1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0,
1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0,
0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0

（序列存储在文件 *L101* 中。）

13. 下列密文是明文通过异或一个 LFSR 输出得到的：

0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0
1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0
0, 1, 1, 1, 0, 1, 0, 1

假设知道明文以下面的序列开始：

1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0

请找出明文（密文存储在文件 *LO11* 中）。

第3章 基础数论

在现代密码体制中，消息在加密和传送之前是以数值的形式表现出来的。加密过程是将输入值转换为输出值的数学运算过程。构建、分析和攻击这些密码体制都需要数学工具，其中最重要的是数论，尤其是有关同余的理论。本章讲述的只是以后几章中所需要的基础，更高级的算法，例如因数分解、离散对数、椭圆曲线等将在以后的章节中介绍。

3.1 基本概念

3.1.1 整除

数论与整数的性质有关，其中最重要的概念之一是整除。

定义：设整数 a 和 b，且 $a\neq 0$，如果存在整数 k 使得 $b=ak$，那么就说 **b 能被 a 整除**，记作 $a\mid b$，或者说 b 是 a 的倍数。

举例：$3\mid 15$，$-15\mid 60$，$7\nmid 18$（不能整除）

下面列举几条关于整除的重要性质。 ■

性质：

(1) 对所有的整数 $a(a\neq 0)$，$a\mid 0$ 和 $a\mid a$ 成立。同理，对任意的整数 b，$1\mid b$ 成立；

(2) 如果 $a\mid b$ 且 $b\mid c$，则 $a\mid c$ 成立；

(3) 如果 $a\mid b$ 且 $a\mid c$，则对所有的整数 s 和 t，$a\mid(sb+tc)$ 成立。

证明：

(1) 因为 $0=a\cdot 0$，所以可以取 $k=0$，由定义 $a\mid 0$ 成立。因为 $a=a\cdot 1$，所以可以取 $k=1$ 使 $a\mid a$ 成立，又因为 $b=b\cdot 1$，所以 $1\mid b$ 成立。

(2) 存在 k 和 l 使得 $b=ak$，$c=bl$ 成立，因此 $c=(kl)a$，所以 $a\mid c$。

(3) 设 $b=ak_1$，$c=ak_2$，那么 $sb+tc=a(sk_1+tk_2)$，所以 $a\mid(sb+tc)$。 □

例如：性质（2）中如果 $a=2$，则有 $2\mid b$，即 b 为偶数，由性质（2）中 $b\mid c$ 可知，偶数 b 的倍数 c 也是偶数（即是 $a=2$ 的整数倍）。

3.1.2　素数

如果整数 p （$p>1$） 只能被 1 或者是它本身整除，而不能够被其他整数整除，则称整数 p 为**素数**（**prime　number**）。

比较小的几个素数是 2，3，5，7，11，13，17，…。一个大于 1 的整数 n 如果不是素数，则称为**合数**（**composite**），即 n 可以表示为整数 a 和 b 的乘积，其中 $1<a$，$b<n$。

众所周知，由欧几里得定理可知存在无穷多的素数，而更为精确的描述是如下定理，该定理在 1896 年已被证明。

素数定理：设 π （x） 是小于 x 的素数的个数，则

$$\pi(x) \approx \frac{x}{\ln x},$$

即当 $x\to\infty$ 时，比值 $\pi(x)/(x/\ln x)\to 1$。

在这里我们不做证明。

它的证明与我们加密的目标相去甚远。在各种应用中，我们需要大的素数，例如 100 位的素数。我们可以估算 100 位的素数的个数：

$$\pi(10^{100}) - \pi(10^{99}) \approx \frac{10^{100}}{\ln 10^{100}} - \frac{10^{99}}{\ln 10^{99}} \approx 3.9\times 10^{97}。$$

因此，素数的数量很多。随后，我们将讨论如何判断一个数是否是素数。

素数是构成整数的因子，每一个整数都是由一个或几个素数的不同次幂相乘得来的。例如 504 和 1125 有如下的因数分解：

$$504 = 2^3 3^2 7, 1125 = 3^2 5^3。$$

并且，这些因数分解都是惟一的，除非颠倒因子的顺序。例如：把 504 因数分解，总是得到三个 2，两个 3，一个 7。如果把素数 41 作为因子就错了。

定理：任何一个正整数都可以分解为几个素数的乘积，且该因数分解是惟一的，除非颠倒因子的顺序。

证明：在证明之前有一个小技巧需要说明，当涉及到乘积的时候，为了方便起见，约定空乘积等于 1，这很类似于所说的 $x^0=1$。因此，正整数 1 可以认为是某个素数的乘积，即空乘积，而每--个素数也可以被认为是该素数的一个乘积因子。

反证法，假设存在正整数，它不能分解为几个素数的乘积，设其中最小的一个为 n，那么 n 不可能为 1(= 空乘积) 或素数(= 只有一个因子的乘积)，因此，n 必然还能够分解。设 $n=ab$，且 $1<a$，$b<n$。由于 n 是最小的正整数，且并不是素数的乘积，a 和 b 都是素数的乘积，但一个素数的乘积乘以另一个素数的乘积仍然是素数的乘积，因此 $n=ab$ 是素数的乘积。这与整数集不是素数的乘积一定是空集相矛盾，因此任何一个正整数都可分解为素数的乘积。

因数分解的惟一性的证明更加复杂，我们需要下述关于素数的重要性质。

补充定理：如果 p 是一个素数，且乘积 ab 能够被 p 整除，那么 $p\mid a$ 或者 $p\mid b$。更一般地，如果乘积 $ab\cdots z$ 能够被素数 p 整除，那么 a，b，...，z 中某个数必能够被 p 整除。

例如，当 $p=2$ 时，如果两个数的乘积为偶数，那么其中必然有一个数为偶数。这个定

理的证明在最后一节给出，在我们讨论欧几里得（Euclidean）算法之后。

下面继续上述定理的证明，假设一个整数 n 可以被因数分解为两种不同的形式：

$$n = p_1^{a_1} p_2^{a_2} \cdots p_s^{a_s} = q_1^{b_1} q_2^{b_2} \cdots q_t^{b_t}$$

其中，底数 $p_1, \ldots, p_s$ 和 $q_1, \ldots, q_t$ 都是素数，指数 a_i 和 b_j 都是非零值。假如一个素数在两个因数分解中都出现，那么将两边同除以这个数。一直这样继续下去，可以假定素数 $p_1, \ldots, p_s$ 中任何一个不是从 q_j 的数中产生。以其中一种因数分解中的素数 p_1 为例，因为 n 能够被 p_1 整除，而 n 等于 $q_1 q_1 \cdots q_1 q_2 \cdots q_t$，面补充定理则证明了 q_j 中的某个数能够被 p_1 整除。因为 q_j 是素数，$p_1 = q_j$。这与原来的假设 p_1 不是从 q_j 的数中产生相矛盾，所以一个整数不可能有两种不同的因数分解，得证。 □

3.1.3 最大公约数（Greatest Common Divisor）

a 和 b 的**最大公约数**是能够同时整除 a 和 b 的最大正整数，记为：$\gcd(a,b)$ 或 (a,b)。在本书中我们使用第一种记法。

例如：$\gcd(6,4) = 2, \gcd(5,7) = 1, \gcd(24,60) = 12$。 ■

如果 $\gcd(a,b) = 1$，我们就说 a 和 b 是**互素的**（**relatively prime**）（也称既约的——译者注）。下面是两种求最大公约数的方法。

1. 如果你能够将 a 和 b 因数分解，则分解之。比较每一个素数在 a 和 b 因数分解中的幂的大小，取较小的一个，将所有素数的幂相乘即可得到 gcd。由下面的例子我们很容易理解：

$$1728 = 2^6 3^3, \quad 135 = 3^3 5, \quad \gcd(1728, 135) = 3^3 = 27$$

$$\gcd(2^5 3^4 7^2, 2^2 5^3 7) = 2^2 3^0 5^0 7^1 = 2^2 7 = 28。$$

注意：由上面的例子我们可以看到，如果一个因数分解中不包含某个素数，那么此素数一定不在最大公约数中。

2. 假设 a 和 b 都是很大的数，那么将其因数分解就不是件容易的事情了。此时，可以通过**欧几里得算法**（**Euclidean algorithm**）来计算 gcd。我们需要回顾一下小学学过的知识：带余数的除法。在对该算法给出精确的叙述之前，先看几个例子。

例：计算 $\gcd(482,1180)$。

解：用 482 去除 1180，商是 2，余数是 216。用 216 去除 482，商是 2，余数是 50。用 50 去除 216，商是 4，余数是 16。用 16 去除 50，商是 3，余数是 2。用 2 去除 16，商是 8，余数是 0。用当前的余数去除上一个余数，一直除下去，直到余数为 0，最后一个非零的余数就是所求的 gcd，本例中 gcd 为 2。

$$\begin{aligned} 1180 &= 2 \cdot 482 + 216 \\ 482 &= 2 \cdot 216 + 50 \\ 216 &= 4 \cdot 50 + 16 \\ 50 &= 3 \cdot 16 + 2 \\ 16 &= 8 \cdot 2 + 0。 \end{aligned}$$

注意数字是如何变化的：

余数 → 除数 → 被除数 → 忽略。

下面是另外一个例子：

$$12345 = 1 \cdot 11111 + 1234$$
$$11111 = 9 \cdot 1234 + 5$$
$$1234 = 246 \cdot 5 + 4$$
$$5 = 1 \cdot 4 + 1$$
$$4 = 4 \cdot 1 + 0。$$

因此，$\gcd(12345, 11111) = 1$。 ■

用上面的例子做向导，我们可以给出欧几里得算法的精确描述。假设 a 大于 b，否则交换 a 和 b，第一步是用 b 去除 a，可以表示为以下形式：

$$a = q_1 b + r_1,$$

如果 $r_1 = 0$，那么 a 能够被 b 整除，最大公约数为 b；如果 $r_1 \neq 0$，则将 b 表示为如下形式：

$$b = q_2 r_1 + r_2。$$

同理，一直进行下去，直到余数为 0，步骤如下：

$$\begin{aligned} a &= q_1 b + r_1 \\ b &= q_2 r_1 + r_2 \\ r_1 &= q_3 r_2 + r_3 \\ \vdots & \quad \vdots \quad \vdots \\ r_{k-2} &= q_k r_{k-1} + r_k \\ r_{k-1} &= q_{k+1} r_k。 \end{aligned}$$

最后的结论是：

$$\gcd(a,b) = r_k。$$

欧几里得算法有以下两个重要的方面：

1. 它不需要因数分解；
2. 速度快。

关于计算 gcd 的准确证明，见习题 17。

由欧几里得算法可以得到下面的重要结论。

定理：设 a 和 b 是两个整数（至少有一个非零），$d = \gcd(a,b)$，则存在整数 x 和 y 使得 $ax + by = d$ 成立。特殊情况下，如果 a 和 b 都是素数，那么存在整数 x 和 y 使得 $ax + by = 1$。

证明：在一般情况下，设 r_j 是欧几里得算法中的余数，则存在 x_j 和 y_j，使得 $r_j = ax_j + by_j$ 成立。初始情况下，$j = 1$，取 $x_1 = 1$，$y_1 = -q_1$，我们可以得出：$r_1 = ax_1 + by_1$。同理，$r_2 = a(-q_2) + b(1 + q_1 q_2)$。假设对于所有的 $i < j$，都有 $r_i = ax_i + by_i$，那么

$$r_j = r_{j-2} - q_j r_{j-1} = ax_{j-2} + bx_{j-2} - q_j(ax_{j-1} + by_{j-1}),$$

重新组合得：

$$r_j = a(x_{j-2} - q_j x_{j-1}) + b(x_{j-2} - q_j y_{j-1})。$$

同理继续计算下去，我们会发现，对于所有的 j 上式都成立（特殊情况下，$j = k$），于是 $r_k = \gcd(a,b)$，由此可证。 □

作为一个推论，我们可以得出因数分解的惟一性证明中所需要的辅助定理。

推论：如果 p 是一个素数，且乘积 ab 能够被 p 整除，那么 $p \mid a$ 或者 $p \mid b$。更一般地，如果乘积 $ab \cdots z$ 能够被素数 p 整除，那么 a，b，...，z 中某个数必能够被 p 整除。

证明：首先证明在 $p \mid ab$ 的情况下，如果 $p \mid a$，即证。如果 $p \nmid a$，只需证明 $p \mid b$。因为 p 是素数，$\gcd(a,p) = 1$ 或 p，又因为 $p \mid a$ 不成立，gcd 不可能为 p，因此，$\gcd(a,p) = 1$，于是存在整数 x 和 y 使得 $ax + py = 1$。两边同乘以 b，可得 $abx + pby = b$。因为 $p \mid ab$ 且 $p \mid p$，所以有 $p \mid (abx + pby)$，因此有 $p \mid b$，证毕。

再证明如果 $p \mid ab\cdots z$，则 $p \mid a$ 或 $p \mid b\cdots z$。如果 $p \mid a$，即证；否则，$p \mid b\cdots z$。此时，我们已将乘数因子的个数减少，同理，或者 $p \mid b$，或者 $p \mid c\cdots z$，……，最后我们得出结论，a，b，...，z 中的某个数必能够被 p 整除，证毕。 □

上述推论中素数的性质仅对素数成立。例如：由 ab 的乘积能够被 6 整除，并不能够推出 a 或 b 能够被 6 整除。例如 $6 = 2 \cdot 3$，设 a 为 2，b 为 3，可见 2 或 3 并不能被 6 整除。$60 = 4 \cdot 15$ 也是同样的道理。一般而言，如果 $n = ab$ 是合数，那么 $n \mid ab$，但是 $n \nmid a$ 和 $n \nmid b$，因此，素数和 1 是惟一符合此推论的整数。

3.2 求解 $ax + by = d$

在欧几里得算法中，我们没有用到商，现在要用到商了。由上一节可知，如果已知两个整数 a 和 b，那么存在整数 x 和 y，使得

$$ax + by = \gcd(a,b)。$$

如何求 x 和 y 呢？假设一开始我们用 a 去除 b，有 $b = q_1 a + r_1$，然后按欧几里得算法继续进行下去。设在 3.1.3 节第一个例子中，接下来的商分别是 q_1，q_2，...q_n，于是有 $q_1 = 2$，$q_2 = 2$，$q_3 = 4, q_4 = 3, q_5 = 8$。形成如下序列：

$$x_0 = 0, x_1 = 1, x_j = -q_{j-1}x_{j-1} + x_{j-2},$$
$$y_0 = 0, y_1 = 1, y_j = -q_{j-1}y_{j-1} + y_{j-2}。$$

形成下列通式：

$$ax_n + by_n = \gcd(a,b)。$$

在第一个例子中，有如下计算：

$$x_0 = 0, x_1 = 1,$$
$$x_2 = -2x_1 + x_0 = -2$$
$$x_3 = -2x_2 + x_1 = 5$$
$$x_4 = -4x_3 + x_2 = -22$$
$$x_5 = -3x_4 + x_3 = 71。$$

同理，可计算出 $y_5 = -29$，用下式表示为：

$$482 \cdot 71 + 1180 \cdot (-29) = 2 = \gcd(482,1180)。$$

由上式可注意到没有用到最后的商，如果用的话，会有 $x_{n+1} = 590$，而 590 正是初始数 1180 除以 gcd（gcd 等于 2）所得的结果。同理，$y_{n+1} = 241$ 是 482/2 所得。

接下来的方法称为**扩展的欧几里得算法（extended Euclidean algorithm）**。它将在下一节中用来解决一些同余问题。

对于一些较小的数，有另外一种方法去求 x 和 y，且此方法不需要用那么多标记符号。

拿上节 gcd(12345,11111) = 1 为例。该方法的思想是用余数 1，4，5，1234 和初始数 11111 和 12345 来计算，最后得到 11111 和 12345 的 gcd 是 1。我们发现：

$$1 = 5 - 1 \cdot 4,$$

因此 1 可以表示为 5 和 4 的线性组合，向上移我们会发现余数 4 可以表示为 1234 和 5 的线性组合，用以下等式表示：

$$4 = 1234 - 246 \cdot 5,$$

所以，

$$1 = 5 - 1 \cdot 4 = 5 - 1 \cdot (1234 - 246 \cdot 5) = 247 \cdot 5 - 1 \cdot 1234,$$

现在使用计算 gcd 的最后两个余数，写出最后未使用的余数（即 5），作为 11111 和 1234 的线性组合，并转换成前面的等式：

$$1 = 247 \cdot (11111 - 9 \cdot 1234) - 1 \cdot 1234 = 247 \cdot 11111 - 2224 \cdot 1234。$$

最后，转换 1234 可得到：

$$1 = 247 \cdot 11111 - 2224 \cdot (12345 - 1 \cdot 11111) = 2471 \cdot 11111 - 2224 \cdot 12345。$$

gcd 的结果 1 是 11111 与 12345 的线性组合，只要 gcd 的计算步骤简洁，动手演算过程就会非常简单。但是，一般而言，先前的算法更加适合于计算机。

3.3 同　余

数论中，最基本且有用的一个概念是模运算或叫同余运算。

定义：设整数 $a,b,n(n \neq 0)$，如果 $a-b$ 是 n 的整数倍（正的或负的），我们就说

$$a \equiv b \pmod n$$

（读作 a 同余于 b 模 n）。

另外一种描述是：如果 a 与 b 的差能够被 n 整除，就说 $a \equiv b \pmod n$，即存在正整数或负整数 k，使得 $a = b + nk$。

举例：

$$32 \equiv 7 \pmod 5;\ -12 \equiv 37 \pmod 7;17 \equiv 17 \pmod{13}。$$ ■

同余与等于非常相似，事实上，人们就是有意这么定义的。

命题：设整数 a，b，c，$n(n \neq 0)$，

(1) 当且仅当 $n \mid a$ 时，$a \equiv 0 \pmod n$。

(2) $a \equiv a \pmod n$。

(3) 当且仅当 $b \equiv a \pmod n$ 时，$a \equiv b \pmod n$。

(4) 如果 $a \equiv b$ 且 $b \equiv c \pmod n$，那么 $a \equiv c \pmod n$。

证明：

(1) $a \equiv 0 \pmod n$ 意味着 $a = a - 0$ 是 n 的倍数，即 $n \mid a$。

(2) $a - a = 0 \cdot n$，所以 $a \equiv a \pmod n$。

(3) 如果 $a \equiv b \pmod n$，记 $a - b = nk$，那么 $b - a = n(-k)$，所以 $b \equiv a \pmod n$；交换 a 和 b 的位置，可以得出相反的结论。

(4) 记 $a = b + nk, c = b + nl$，那么 $a - c = n(k - l)$，所以 $a \equiv c \pmod n$。 □

通常需要处理整数模 n 运算，记为 Z_n，这些整数可以当作是集合 $\{0, 1, 2, \cdots, n-1\}$。此外，还有减法和乘法的模 n 运算。如果 a 是任意整数，在该集合上，可以用如下形式表示 a 除以 n 得到的余数：

$$a = nq + r, \qquad 0 \leqslant r < n$$

（这仅适用于有余数的除法；q 是商而 r 是余数）。于是 $a \equiv r(\bmod n)$，则每一个数 a 同余于 r 模 n 对 $0 \leqslant r < n$ 中的所有整数 r 都成立。

命题：设整数 a, b, c, d, $n(n \neq 0)$，假设 $a \equiv b(\bmod n)$，且 $c \equiv d(\bmod n)$，那么

$$a + c \equiv b + d, a - c \equiv b - d, ac \equiv bd(\bmod n)。$$

证明：

设存在整数 k 和 l，使得 $a = b + nk, c = d + nl$，那么 $a + c = b + d + n(k + l)$，因此 $a + c \equiv b + d\ (\bmod n)$。同理可证：$a - c \equiv b - d$。对于乘法，有 $ac = bd + n(dk + bl + nkl)$，所以 $ac \equiv bd$。

□

这个命题说明可以运用同余进行加减乘运算，但是要注意，在进行除运算的时候，应非常小心。

如果要对两个数的乘积进行模 n 运算，我们首先要相乘之后取整，如果结果小于 n，就停止计算，如果结果大于 $n-1$，就用 n 去除，然后取余。用上述方法也可以进行加减运算，例如，整数模 6 的结果就有以下的加法表格：

+	0	1	2	3	4	5
0	0	1	2	3	4	5
1	1	2	3	4	5	0
2	2	3	4	5	0	1
3	3	4	5	0	1	2
4	4	5	0	1	2	3
5	5	0	1	2	3	4

整数模 6 的结果有以下的乘法表格：

×	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	2	3	4	5
2	0	2	4	0	2	4
3	0	3	0	3	0	3
4	0	4	2	0	4	2
5	0	5	4	3	2	1

举例：下面这个例子将告诉我们如何进行模 n 的代数运算。考虑这个问题：解 $x + 7 \equiv 3(\bmod 17)$

解：$x \equiv 3 - 7 \equiv -4 \equiv 13(\bmod 17)$。 ■

答案为负数并没有错，但是，通常情况下，我们将用 0 到 $n-1$ 之间的整数来表示模 n

运算的结果。

3.3.1 除法

模 n 除法比传统的除法要有一些技巧，通常的法则是：在 $\gcd(a,n)=1$ 的情况下用 $a \pmod n$ 去除。

命题：设整数 $a,b,c,n(n\neq 0)$，且 $\gcd(a,n)=1$，如果 $ab\equiv ac \pmod n$，那么 $b\equiv c \pmod n$；换句话说，如果 a 和 n 是互素的，我们可以在同余式两边同除以 a。

证明：因为 $\gcd(a,n)=1$，所以存在整数 x，y 使得 $ax+ny=1$，两边同乘以 $b-c$ 得：

$$(ab-ac)x+n(b-c)y=b-c,$$

因为 $ab-ac$ 是 n 的倍数，$n(b-c)y$ 也是 n 的倍数，所以 $b-c$ 是 n 的倍数，即 $b\equiv c \pmod n$。 □

例：解 $2x+7\equiv 3 \pmod{17}$。

解：$2x\equiv 3-7\equiv -4$，于是有 $x\equiv -2\equiv 15 \pmod{17}$。因为 $\gcd(2,17)=1$，所以能够被 2 整除。 ■

例：解 $5x+6=13 \pmod{11}$。

解：$5x\equiv 7 \pmod{11}$。现在怎么办？我们希望能够被 5 整除，但是，7/5 意味着模 11？注意到 $7\equiv 18\equiv 29\equiv 40\equiv \cdots \pmod{11}$，因此 $5x\equiv 7$ 等价于 $5x\equiv 40$。

现在，两边同除以 5，可以得到结果 $x\equiv 8 \pmod{11}$。注意：$7\equiv 8\cdot 5 \pmod{11}$，所以 8 与 7/5 的作用类似。 ■

后一个例子也可以用另外一种方法。

因为 $5\cdot 9\equiv 1 \pmod{11}$，我们可以发现，9 是 $5 \pmod{11}$ 的乘法逆元素，所以两边同除以 5 等价于两边同乘以 9。如果我们要求 $5x\equiv 7 \pmod{11}$，可以将两边同乘以 9，得到：

$$x\equiv 45x\equiv 63\equiv 8 \pmod{11}$$

命题：假设 $\gcd(a,n)=1$，则存在整数 s，t，使得 $as+nt=1$（由扩展的欧几里得算法可知），那么 $as\equiv 1 \pmod n$，因此 s 是 $a \pmod n$ 的乘法逆元素。

证明：$\because\ as-1=-nt,$

$\therefore\ as-1$ 是 n 的整数倍。 □

扩展的欧几里得算法在计算 $a \pmod n$ 的乘法逆元素时十分有效。

例：解 $11111x\equiv 4 \pmod{12345}$。

解：参照前面例子中计算 $\gcd(12345,11111)$，我们得到商 $q_1=1, q_2=9, q_3=246, q_4=1, q_5=4$，因此，在扩展的欧几里得算法中，$x_0=0, x_1=1, x_2=-1, x_3=10, x_4=-2461, x_5=2471$，我们可以得出：$11111\cdot 2471+12345\cdot y_5=1$，因此，

$$11111\cdot 2471\equiv 1 \pmod{12345},$$

在原来的同余式两边同乘以 2471，可以得到结果：

$$x\equiv 9884 \pmod{12345}。$$

事实上，这意味着，我们在计算模 12345 时，如果遇到了 4/11111，可以用 9884 来代替。这看起来似乎有点奇怪，但是想一想 4/11111 是什么意思，它只是一个量的标志，当乘 11111 的时候得到 4。当我们计算模 12345 的时候，9884 也有同样的性质，因为 11111 ×

$9884 \equiv 4 \pmod{12345}$。 ■

下面总结出一些结论。

3.3.2 求 $a^{-1} \pmod{n}$

1. 用扩展的欧几里得算法求出满足条件 $as + nt = 1$ 的整数 s 和 t。
2. $a^{-1} \equiv s \pmod{n}$。

3.3.3 当 $\gcd(a,n) = 1$ 时，解 $ax \equiv c \pmod{n}$

（或者，在 $\gcd(a,n) = 1$ 的条件下，对 c/a 进行模 n 运算。）
1. 用扩展的欧几里得算法求出满足条件 $as + nt = 1$ 的整数 s 和 t。
2. 解 $x \equiv cs \pmod{n}$（等价于用 $cs \pmod{n}$ 代替 c/a）。

3.3.4 如果 $\gcd(a,n) > 1$ 怎么办

有时候，我们需要求解 $\gcd(a,n) = d > 1$ 时 $ax \equiv b \pmod{n}$ 的同余式，其求解过程如下：

1. 如果 b 不能够被 d 整除，无解。
2. 假设 $d \mid b$，考虑下面的同余式：

$$(a/d)x \equiv b/d \pmod{n/d},$$

注意到 a/d，b/d，n/d 是整数，且 $\gcd(a/d, n/d) = 1$，用上面的方法解这个同余式可以得到一个解 x_0。

3. 原同余式 $ax \equiv b \pmod{n}$ 的解为：

$$x_0, x_0 + (n/d), x_0 + 2(n/d), \ldots, x_0 + (d-1)(n/d) \qquad \pmod{n}。$$

举例： 解 $12x \equiv 21 \pmod{39}$。

解：$\gcd(12,39) = 3$，两边同除以 3，可以得到新的同余式：$4x \equiv 7 \pmod{13}$。用试探法或者是用扩展的欧几里得算法，我们可以得到一个解 $x = 5$，因此原同余式的解为 $x \equiv 5, 18, 31 \pmod{39}$。 ■

前面的方程式都是关于 x 的一次方的，但是对非线性同余式也非常的有用。在本书中的许多地方，我们可以看到下面的方程式：

$$x^2 \equiv a \pmod{n}。$$

首先，考虑方程式 $x^2 \equiv 1 \pmod{7}$，当把值 0，1，2，3，4，5，6 代入 x 时，会发现解为 $x \equiv 1, 6 \pmod{7}$。通常情况下，当 p 是奇素数时，$x^2 \equiv 1 \pmod{p}$ 只有两个解 $x \equiv \pm 1 \pmod{p}$（见练习 4）。

现在，考虑同余方程式 $x^2 \equiv 1 \pmod{15}$。当我们把值 0，1，2，...，14 代入 x 时，会发现解为 $x = 1, 4, 11, 14$。例如：$11^2 \equiv 121 \equiv 1 \pmod{15}$；因此，二次同余方程式的复合模数有两个或两个以上的解。相对而言，二次同余实数方程等式最少有两个以上的解。在 3.4 节，我们将要讨论这种现象。在 6.4 节、11.1 节、12.2 节还将遇到它的应用。

3.3.5 分数的计算

在大多数情况下，分数的模 n 运算非常简单。例如：1/2(mod 12345) 比 6173(mod 12345) 要容易得多（注意：$2\times6173\equiv1(\bmod 12345)$）。通常情况下的原则是：当 $\gcd(a,n)=1$ 时，分数 b/a 可以进行模 n 运算。当然，必须要说明的是，$b/a(\bmod n)$ 的意思是 $a^{-1}b(\bmod n)$，这里 a^{-1} 意味着模 n 运算满足 $a^{-1}a\equiv1(\bmod n)$。如果它被当作分数看待，就不会出错。

也可以这样认为，"1/2" 仅仅是具有如下性质的一个标志：如果 1/2 乘以 2，将得到 1。在所有的计算中都包含着 "1/2"，但是，我们只是运用了它的性质。当我们进行模 12345 运算时，6173 也具有这样的性质，因为 $6173\times2\equiv1(\bmod 12345)$；因此 1/2 (mod 12345) 与 6173(mod 12345) 是可以互换的。

为什么不能运用分数进行呢？当然我们不能计算 1/6(mod 6)，因为那就意味着被 0(mod 6)除，但是即便是在计算 1/2(mod 6) 时，也是有麻烦的。例如：计算 $2\equiv8(\bmod 6)$，我们不能两边同乘以 1/2，因为 $1\not\equiv4(\bmod 6)$。根本问题在于 $\gcd(2,6)=2\neq1$，由于 2 是 6 的一个因子，我们可以把两边同除以 2 理解为 "除以 0"，在任何情况下都是不允许这么做的。

3.4 中国剩余定理

译者注：在中国数学史上，广泛流传着一个"韩信点兵"的故事：韩信是汉高祖刘邦手下的大将，他英勇善战，智谋超群。据说韩信的数学水平非常高超，他在点兵的时候，为了保住军事机密，不让敌人知道自己部队的实力，先令士兵从 1 至 3 报数，然后记下最后一个士兵所报之数；再令士兵从 1 至 5 报数，也记下最后一个士兵所报之数；最后令士兵从 1 至 7 报数，又记下最后一个士兵所报之数；这样，他很快就算出了自己部队士兵的总人数，而敌人则始终无法弄清他的部队究竟有多少名士兵。这个故事中所说的韩信点兵的计算方法，就是现在被称为"中国剩余定理"的一次同余式解法。它是中国古代数学家的一项重大创造，在世界数学史上具有重要的地位。

在大多数情况下，把一个同余式的模 n 运算分解为模 n 因子的同余式的运算将非常简单。下面举例说明：假设一个数 x 满足 $x\equiv25(\bmod 42)$，即存在整数 k 使得 $x=25+42k$，将 42 因数分解，$42=7\cdot6$，我们得到 $x=25+7(6k)$，表明 $x\equiv25\equiv4(\bmod 7)$，同理，可得到 $x=25+6(7k)$，有 $x\equiv25\equiv1(\bmod 6)$，因此，

$$x\equiv25(\bmod 42)\Rightarrow\begin{cases}x\equiv4(\bmod 7)\\x\equiv1(\bmod 6)\end{cases}。$$

中国剩余定理证实了这个过程是可以反过来的。即在一定条件下，一个同余体系可以用某条件下的一个单一的同余式来表示。

中国剩余定理：假设 $\gcd(m,n)=1$，给定 a 和 b，对同余方程组

$$x\equiv a\ (\bmod m),\ x\equiv b\ (\bmod n),$$

存在惟一的解为 $x(\bmod mn)$。

证明：存在整数 s,t 使得 $ms+nt=1$ 成立，因为 $ms\equiv 1(\bmod n)$，$nt\equiv 1(\bmod m)$。设 $x=bms+ant$，那么 $x\equiv ant\equiv a(\bmod m)$，且 $x\equiv bms\equiv b(\bmod n)$。假设 x_1 是另外一个解，则有 $x\equiv x_1(\bmod m)$，且 $x\equiv x_1(\bmod n)$，所以 $x-x_1$ 既是 m 的倍数，也是 n 的倍数。

推论：设整数 m,n 且 $\gcd(m,n)=1$，假如整数 c 是 m 的倍数，也是 n 的倍数，那么 c 是 mn 的倍数。

证明：设 $c=mk=nl$。存在整数 s、t，使得 $ms+nt=1$，两边同乘以 c，则 $c=cms+cnt=mnls+mnkt=mn(ls+kt)$。得证。 □

为了完成中国剩余定理的证明，设 $c=x-x_1$，由此可知 $x-x_1$ 是 mn 的倍数。因此，$x\equiv x_1(\bmod mn)$。这说明了这个方程组的任意两个解对模 mn 都是同余的，得证。

例：解 $x\equiv 3\ (\bmod 7)$，$x\equiv 5\ (\bmod 15)$。 □

解：$x\equiv 80\ (\bmod 105)$（注：$105=7\cdot 15$）。因为 $80\equiv 3(\bmod 7)$ 和 $80\equiv 5(\bmod 15)$，80 是一个解；这个定理保证了这个解的存在性，并且由 mn 的乘积惟一决定，本例中这个乘积就是 105。

■

如何来求解呢？第一种方法，对于比较小的数 m，n 而言，列出所有的与 $b(\bmod n)$ 同余的数，直至找到与 $a(\bmod m)$ 同余的数。例如，与 $5(\bmod 15)$ 同余的数是

$$5,20,35,50,65,80,95,\ldots\ldots$$

对于 mod 7，同余的数是 5，6，0，1，2，3，4，…… 因为我们需要的是 $3(\bmod 7)$，所以我们选择 80。

对于稍大的数 m，n，用列举法效率很低，但是有一种类似的方法可以解决这个问题。与 $b\ (\bmod n)$ 同余的数实际上是 $b+nk$，k 为整数，因此，我们需要解 $b+nk\equiv a\ (\bmod m)$，即

$$nk\equiv a-b\ (\bmod m)。$$

因为 $\gcd(m,n)=1$，存在 $n(\bmod m)$ 的乘法逆元素 i，两边同乘以 i，得

$$k\equiv (a-b)i\ (\bmod m)。$$

代入 $x=b+nk$，即可得到结果。

对于大的整数，余数定理的证明给出了如上所述的求解 x 的有效方法。

例：解 $x\equiv 7\ (\bmod 12345)$，$x\equiv 3(\bmod 11111)$。

解：由 3.3 节的计算可知，$11111(\bmod 12345)$ 的逆元素 $i=2471$，因此，$k\equiv 2471(7-3)\equiv 9884\ (\bmod 12345)$，结果是 $x=3+11111\cdot 9884\equiv 109821127(\bmod(11111\cdot 12345))$。■

如何运用中国剩余定理呢？主要的思想是要计算出对合数求模的同余式，首先要把它分解为几个素数的乘积，然后将结果重新组合，即可得到结果。这种做法的优点是对素数或素数的幂的运算要比对合数的运算简单得多。

假设要解 $x^2\equiv 1\ (\bmod 35)$，而 $35=5\cdot 7$，有

$$x^2\equiv 1\ (\bmod 35)\Leftrightarrow\begin{cases}x^2\equiv 1\ (\bmod 7)\\ x^2\equiv 1\ (\bmod 5)。\end{cases}$$

现在，$x^2\equiv 1\ (\bmod 5)$ 有两个解：$x\equiv\pm 1\ (\bmod 5)$。同理，$x^2\equiv 1\ (\bmod 7)$ 也有两个解：$x\equiv\pm 1\ (\bmod 7)$。我们可将其写成如下 4 种形式：

$$x\equiv 1\ (\bmod 5),x\equiv 1\ (\bmod 7)\rightarrow x\equiv 1\ (\bmod 35),$$

$$x \equiv 1 \pmod 5, x \equiv -1 \pmod 7 \rightarrow x \equiv 6 \pmod{35},$$
$$x \equiv -1 \pmod 5, x \equiv 1 \pmod 7 \rightarrow x \equiv 29 \pmod{35},$$
$$x \equiv -1 \pmod 5, x \equiv -1 \pmod 7 \rightarrow x \equiv 34 \pmod{35}。$$

因此 $x^2 \equiv 1 \pmod{35}$ 的解为：$x \equiv 1,6,29,34 \pmod{35}$。

通常情况下，如果 $n = p_1p_2\cdots p_r$ 是 r 个不同奇素数的乘积，那么 $x^2 \equiv 1 \pmod n$ 有 2^r 个不同的解。这由下列定理可知。

中国剩余定理（一般形式）：设 $m_1, m_2, \cdots\cdots, m_k$ 是整数，且对任意的 $i, j(i \neq j)$，$\gcd(m_i, m_j) = 1$，给定整数 a_1，a_2，……，a_k，对下列所有同余方程组，

$$x \equiv a_1 \pmod{m_1}, x \equiv a_2 \pmod{m_2}, \ldots\ldots, x \equiv a_k \pmod{m_k},$$

存在惟一个解 $x \pmod{m_1\cdots m_k}$。

例如：该定理使得下列同余方程组有解

$$x \equiv 1 \pmod{11}, x \equiv -1 \pmod{13}, x \equiv 1 \pmod{17},$$

事实上，$x \equiv 1871 \pmod{11 \cdot 13 \cdot 17}$ 是所求的解。

用该定理求解 x 的过程见练习 13。

3.5 模的幂计算

在本书中，我们将讨论下列形式的计算：

$$x^a \pmod n。$$

在本节和随后的几节中，我们将讨论指数的模运算。

假如要计算 $2^{1234} \pmod{789}$。如果我们先计算 2^{1234}，然后再进行模运算，将会遇到很大的数，而最后的结果却只有 3 位数。若要先乘，然后取余，计算 2 的 1234 次方至少要乘 1233 次，这种方法太慢了，所以实际应用中并不可取，尤其是在指数很大的时候。

下面介绍一种更有效的方法（所有同余式均是模 789 的）。

我们先计算 $2^2 \equiv 4 \pmod{789}$，然后两边平方，得到下列同余式：

$$2^4 \equiv 4^2 \equiv 16$$
$$2^8 \equiv 16^2 \equiv 256$$
$$2^{16} \equiv 256^2 \equiv 49$$
$$2^{32} \equiv 34$$
$$2^{64} \equiv 367$$
$$2^{128} \equiv 559$$
$$2^{256} \equiv 37$$
$$2^{512} \equiv 580$$
$$2^{1024} \equiv 286$$

因为 $1234 = 1024 + 128 + 64 + 16 + 2$（这也说明了 1234 等于二进制的 10011010010）。所以，

$$2^{1234} \equiv 286 \cdot 559 \cdot 367 \cdot 49 \cdot 4 \equiv 481 \pmod{789}。$$

由上可知，我们所计算的数最大也不会超过 788^2。

同样，一般情况下，如果我们要计算 $a^b \pmod n$，最多只需计算 $2\log_2(b) \pmod n$，并且不会遇到比 n^2 更大的数。这也就是说，模的幂运算可以快速地完成，并且不需要太大的内存。

当 a，b，n 是 100 位的数时，这种算法的效率很高。如果我们先计算 a^b，然后再进行模 n 运算，那么计算机的内存会溢出，因为 a^b 有多于 10 的 100 次方的位数。然而，用刚才介绍的方法计算 $a^b \pmod n$，在 700 步之内即可完成，且计算中根本不会遇到多于 200 位的数。

该算法的过程见习题 12。

3.6 费尔马小定理和欧拉定理

数论中两个最基本的定理是费尔马小定理和欧拉定理。这两个定理一直以来都具有重要的理论价值，近期人们发现它们在密码学中还具有重要的应用，本书中将反复用到这两个定理。

费尔马小定理：如果 p 是一个素数，且 p 不能够被 a 整除，那么

$$a^{p-1} \equiv 1 \pmod p。$$

证明：设 $S = \{1,2,3,...,p-1\}$。

考虑映射 $\psi: S \to S$，它是由 $\psi(x) = ax \pmod p$ 定义的。例如：当 $p=7$，$a=2$ 时，映射 ψ 将 x 乘以 2，然后进行模 7 运算。

我们需要检验的是，如果 $x \in S$，那么 $\psi(x)$ 是否在 S 里，即 $\psi(x) \neq 0$。假设 $\psi(x)=0$，那么 $ax \equiv 0 \pmod p$。因为 $\gcd(a,p)=1$，我们可以将两边同除以 a，得到 $x \equiv 0 \pmod p$，所以 $x \notin S$。这说明 $\psi(x) \neq 0$，因此 $\psi(x) \in S$。现在假设存在 x，$y \in S$，使得 $\psi(x) = \psi(y)$。即 $ax \equiv ay \pmod p$，因为 $\gcd(a,p)=1$，两边同除以 a，得到 $x \equiv y \pmod p$。所以，如果 x 和 y 不同，那么 $\psi(x)$ 和 $\psi(y)$ 也是不同的。因此，

$$\psi(1), \psi(2), \psi(3), \cdots, \psi(p-1)$$

是 S 中的不同元素。由于 S 中只有 $p-1$ 个元素，所以 S 中的元素必有序。它的顺序如下：

$$\begin{aligned}
&1 \cdot 2 \cdot 3 \cdots (p-1) \\
&\equiv \psi(1) \cdot \psi(2) \cdot \psi(3) \cdots \psi(p-1) \\
&\equiv (a \cdot 1)(a \cdot 2)\ (a \cdot 3)\ \cdots\ (a \cdot (p-1)) \\
&\equiv a^{p-1}(1 \cdot 2 \cdot 3 \cdots (p-1))\ \pmod p。
\end{aligned}$$

因为对所有的 $j \in S$，都有 $\gcd(j,p)=1$，两边同除以 1，2，3，...，p－1，所以最后结果是 $1 \equiv a^{p-1} \pmod p$。证毕。 □

例如：$2^{10} = 1024 \equiv 1 \pmod{11}$。由此，我们可以计算 $2^{53} \pmod{11}$：$2^{53} = (2^{10})^5 2^3 \equiv 1^5 2^3 \equiv 8 \pmod{11}$。注意，虽然是计算模 11 的情况，但我们的基本计算使用的是模 10 而不是模 11，换句话说，即由 $53 \equiv 3 \pmod{10}$，可得出 $2^{53} \equiv 2^3 \pmod{11}$。 ■

通常情况下，如果 $2^{n-1} \equiv 1 \pmod n$，那么 n 为素数。然而，也有例外：$561 = 3 \cdot 11 \cdot 17$ 是合数，但是 $2^{560} \equiv 1 \pmod{561}$。看如下计算：因为 $560 \equiv 0 \pmod 2$，所以 $2^{560} \equiv 2^0 \equiv$

1(mod 3)。同理，因为 $560 \equiv 0 \pmod{10}$，且 $560 \equiv 0 \pmod{16}$，所以 $2^{560} \equiv 1 \pmod{11}$ 且 $2^{560} \equiv 1 \pmod{17}$。由中国剩余定理可知，$2^{560} \equiv 1 \pmod{561}$。

另外一个特例是：$1729 = 7 \cdot 13 \cdot 19$，但是这样的特例在实际中并不常见。因此，如果 $2^{n-1} \equiv 1 \pmod n$，那么 n 很有可能为素数。当然，如果 $2^{n-1} \not\equiv 1 \pmod n$，那么 n 不可能为素数。由于 $2^{n-1} \pmod n$ 的计算很快（见 3.5 节），这就给我们提供了一种寻找素数的方法。即选择初始的 n_0，然后依次检验接下来的奇数 $n \geqslant n_0$。看是否满足 $2^{n-1} \equiv 1 \pmod n$，如果 n 不满足，则进行下一个 n 的检验；如果 n 满足，则用更复杂的方法检验（见 6.3 节）。这种方法的优点是：其速度要比因数分解快得多，尤其是它可以排除许多 n。当然，也有一些方法可以加速寻找过程，例如，首先排除那些有小素数因子的 n。

当然我们也需要对合数模 n 的类似的费尔马定理。设 $\phi(n)$ 是整数 $1 \leqslant a \leqslant n$ 中满足 $\gcd(a,n) = 1$ 的个数。比如 $n = 10$，有 4 个这样的整数，即 1，3，7，9，所以 $\phi(10) = 4$。通常 ϕ 叫做**欧拉 ϕ - 函数**。

如果 p 是一个素数且 $n = p^r$，那么为了保证 $\gcd(a,n) = 1$，我们必须移去第 p 个数字，可以得到：

$$\phi(p^r) = \left(1 - \frac{1}{p}\right)p^r,$$

特殊情况下，

$$\phi(p) = p - 1,$$

更一般地，由中国剩余定理可知，对任意 n，有

$$\phi(n) = n\prod_{p \mid n}\left(1 - \frac{1}{p}\right),$$

这里的乘积是不同素数 p 整除 n 的积。当 $n = pq$ 是两个不同的素数的乘积时，结果为：

$$\phi(pq) = (p-1)(q-1)。$$

例如：

$$\phi(10) = (2-1)(5-1) = 4$$

$$\phi(120) = 120\left(1 - \frac{1}{2}\right)\left(1 - \frac{1}{3}\right)\left(1 - \frac{1}{5}\right) = 32$$ ■

欧拉定理： 如果 $\gcd(a,n) = 1$，那么

$$a^{\phi(n)} \equiv 1 \pmod n。$$

证明：此定理的证明和费尔马小定理的证明类似。设 S 是一个整数集合，对 $1 \leqslant x \leqslant n$，有 $\gcd(x,n) = 1$，设 ψ：$S \to S$ 定义成 $\psi(x) \equiv ax \pmod n$。同费尔马定理的证明，对所有 $x \in S$，$\psi(x)$ 就是集合 S 中按照一定顺序排列的数。因此，

$$\prod_{x \in S} x \equiv \prod_{x \in S} \Psi(x) \equiv a^{\phi(n)} \prod_{x \in S} x,$$

两边除以因子 $x \in S$，左边可以得到：$1 \equiv a^{\phi(n)} \pmod n$。证毕。 □

注意，我们可以发现，当 $n = p$ 是素数的时候，欧拉定理和费尔马小定理相同。

例： 求 7^{803} 后三位数字。

解：易知后三位数字和 mod 1000 的结果一样。因为 $\phi(1000) = 1000(1 - \frac{1}{2})(1 - \frac{1}{5}) = 400$，有 $7^{803} = (7^{400})^2 7^3 \equiv 7^3 \equiv 343 \pmod{1000}$，所以后三位数字为 343。

在本例中，可以改变指数从 803 到 3，因为 $803 \equiv 3 \pmod{\phi(1000)}$。 ■

例：计算 $2^{43210} \pmod{101}$。

解：由费尔马定理，可以得到：$2^{100} \equiv 1 \pmod{101}$，所以

$$2^{43210} \equiv (2^{100})^{432} 2^{10} \equiv 1^{432} 2^{10} \equiv 1024 \equiv 14 \pmod{101}。$$

在这种情况下，我们可以将指数 43210 换为 10，因为 $43210 \equiv 10 \pmod{100}$。 ■

总而言之，我们可以得到下述结论。

基本原理：设 a，n，x，y 都是整数，$n \geqslant 1$，且 $\gcd(a,n) = 1$，如果 $x \equiv y \pmod{\phi(n)}$，那么 $a^x \equiv a^y \pmod{n}$。换句话说，如果要计算 mod n，应该先计算 mod$\phi(n)$。

证明：记 $x = y + \phi(n)k$，那么

$$a^x = a^{y+\phi(n)k} = a^y(a^{\phi(n)})^k \equiv a^y 1^k \equiv a^y \pmod{n},$$

证毕。 □

复习本书前面所讲的余数定理很有必要，回顾前面的例子，直到你证实指数模 $400 = \phi(1000)$，而模 100 又是什么（即并不是许多人认为的那样，在这些例子中他们错误地使用指数模 1000 和模 101）。

3.7 本 原 根

考虑下面 3 (mod 7) 的幂运算：

$$3^1 \equiv 3,\ 3^2 \equiv 2,\ 3^3 \equiv 6,\ 3^4 \equiv 4,\ 3^5 \equiv 5,\ 3^6 \equiv 1。$$

注意我们得到了所有非零的模 7 的同余类是 3 的幂次方。这就是说，3 是模 7 的本原根（用术语乘法生成器可能更好，但不太常用）。同理，每一个非零的模 13 同余类都是 2 的幂，因此 2 是模 13 的本原根。但是，$3^3 \equiv 1 \pmod{13}$，所以只有 1，3，9 是 3 的幂，因此 3 不是模 13 的本原根。模 13 的本原根是 2，6，7，11。

一般来说，当 p 是素数时，**模 p 本原根（primitive root）**[1] 是一个数，它的幂产生每一个非零的模 p 的类。存在 $\phi(p-1)$ 个模 p 的本原根，特殊情况下，至少总有一个。实际上，求一个根并不难，尤其是已知 $p-1$ 的因数分解的时候。见练习 10。

下面的命题概括了有关本原根的要点。

命题：设对素数 p 而言，g 是一个本原根。

1. 如果 n 是整数，那么 $g^n \equiv 1 \pmod{p}$ 当且仅当 $n \equiv 0 \pmod{p-1}$。
2. 如果 j 和 k 都是整数，那么 $g^j \equiv g^k \pmod{p}$ 当且仅当 $j \equiv k \pmod{p-1}$。

证明：如果 $n \equiv 0 \pmod{p-1}$，那么存在 m，使得 $n = (p-1)m$。因此，由费尔马定理可知，

$$g^n \equiv (g^m)^{p-1} \equiv 1 \pmod{p}$$

相反，假设 $g^n \equiv 1 \pmod{p}$，我们需要证明 n 能够被 $p-1$ 整除，因此用 $p-1$ 去除 n，则只需要证明余数为 0 即可。记

$$n = (p-1)q + r, \qquad (0 \leqslant r < p-1)$$

---

1 译者注：对本原根的叫法有些书称本原元，还有称原根的，本书一律称本原根。

（这里仅用商 q 和余数 r 表示除），有

$$1 \equiv g^n \equiv (g^q)^{p-1} g^r \equiv 1 \cdot g^r \equiv g^r \pmod p。$$

假设 $r>0$，如果我们考虑 $g \pmod p$ 的幂 $g, g^2, \ldots$ 经过 r 步之后，得：

$$g^{r+1} \equiv g, g^{r+2} \equiv g^2, \ldots$$

因此，$g \pmod p$ 的幂仅产生 r 个数 g，g^2，…，1。既然 r 小于 $p-1$，并不是每一个模 p 的结果都是 g 的幂。这就同 g 是本原根的假设相矛盾。

惟一的可能是 r 等于 0。这就意味着 $n=(p-1)r$，因此 n 可以被 $p-1$ 整除。这就证明了第一部分。

对于第二部分，假设 $j \geq k$（如果不是，则交换 j 和 k）。如果 $g^j \equiv g^k \pmod p$，用 g^k 同时除等式两边得 $g^{j-k} \equiv 1 \pmod p$。由第一部分，$j-k \equiv 0 \pmod{p-1}$，因此 $j \equiv k \pmod{p-1}$。反之亦然。对 $j \equiv k \pmod{p-1}$，那么 $j-k \equiv 0 \pmod{p-1}$，因此 $g^{j-k} \equiv 1 \pmod p$，由第一部分，乘以 g^k 即可产生结果。证毕。 □

3.8 模 n 逆矩阵

只要应用 3.3 节中处理小数部分的规则，通过一般的矩阵求逆就可以实现模 n 矩阵 r 的逆运算。我们需要的一个基本事实是：一个平方矩阵是模 n 可逆的当且仅当它的行列式和 n 是互素的。

在这里我们指的只是小矩阵，因为这些是本书中的例子所需要的。这样，求一个矩阵的逆矩阵最简单的方法就是用有理数，然后再转成模 n 的数。通常认为一个整数矩阵的逆可以由另一个矩阵除以原矩阵的行列式得到。因为假设行列式和 n 值是互素的，我们可以按 3.3 节的内容来转换行列式。

例如，举个 2 × 2 的例子，通常的公式是：

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}^{-1} = \frac{1}{ad-bc}\begin{pmatrix} d & -b \\ -c & a \end{pmatrix},$$

为此我们要找到 $ad-bc \pmod n$ 的逆。

例：假如我们要求 $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \pmod{11}$ 的逆。因为 $ad-bc=-2$，我们需要求 $-2 \bmod 11$ 的逆。因为 $5 \times (-2) \equiv 1 \pmod{11}$，所以可以用 5 替换 $-1/2$，然后得到：

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}^{-1} \equiv \frac{-1}{2}\begin{pmatrix} 4 & -2 \\ -3 & 1 \end{pmatrix} \equiv 5\begin{pmatrix} 4 & -2 \\ -3 & 1 \end{pmatrix} \equiv \begin{pmatrix} 9 & 1 \\ 7 & 5 \end{pmatrix} \pmod{11}$$

一种快速计算如下：

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}\begin{pmatrix} 9 & 1 \\ 7 & 5 \end{pmatrix} = \begin{pmatrix} 23 & 11 \\ 55 & 23 \end{pmatrix} \equiv \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \pmod{11}$$ ■

例：假如我们要求如下矩阵的逆

$$M = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 4 & 9 \end{pmatrix} \pmod{11}$$

行列式是 2 且 M 的有理逆矩阵是:

$$\frac{1}{2}\begin{pmatrix} 6 & -5 & 1 \\ -6 & 8 & -2 \\ 2 & -3 & 1 \end{pmatrix}$$

(有关其他求逆矩阵的方法，可参考线性代数。) 我们可以用 6 模 11 来代替 1/2，得:

$$M^{-1} \equiv \begin{pmatrix} 3 & 3 & 6 \\ 8 & 4 & 10 \\ 1 & 4 & 6 \end{pmatrix} \pmod{11}。$$

■

为什么行列式和 n 必须是互素的呢? 假如 $MN \equiv I \pmod{n}$，I 是单位矩阵，那么

$$\det(M)\det(N) \equiv \det(MN) \equiv \det(I) \equiv 1 \pmod{n},$$

因此，$\det(M)$ 有一个模 n 的逆元素，这意味着 $\det(M)$ 和 n 必须是互素的。

3.9 模 n 平方根

假如已知 $x^2 \equiv 71 \pmod{77}$ 有一个解。怎样才能找到它的一个解呢? 又如何找到所有的解呢? 更常见的是找出方程 $x^2 \equiv b \pmod{n}$ 所有的解，这里 $n = pq$ 是两个素数的乘积。一旦 n 的因数分解已知的话，下面我们会展示这相当简单。相反，如果知道了所有的解，同样因数分解 n 也就非常简单了。

首先考虑模素数 p 平方根的情况。最简单的情况是 $p \equiv 3 \pmod{4}$，这正是我们所要讨论的。这种情况下当 $p \equiv 1 \pmod{4}$ 时比较复杂。见 [Cohen, pp. 31-34]。

命题: 设 $p \equiv 3 \pmod{4}$ 是素数，y 是一个整数。令 $x \equiv y^{(p+1)/4} \pmod{p}$。

1. 如果 y 有一个模 p 平方根，那么 y 模 p 的平方根是 $\pm x$。

2. 如果 y 没有模 p 平方根，那么 $-y$ 有一个模 p 平方根，并且 $-y$ 模 p 的平方根是 $\pm x$。

证明: 如果 $y \equiv 0 \pmod{p}$，那么所有的参数都是可忽略的，所以假设 $y \not\equiv 0 \pmod{p}$。由费尔马定理得到 $y^{p-1} \equiv 1 \pmod{p}$，因此，

$$x^4 \equiv y^{p+1} \equiv y^2 y^{p-1} \equiv y^2 \pmod{p},$$

这就说明: $(x^2+y)(x^2-y) \equiv 0 \pmod{p}$，所以 $x^2 \equiv \pm y \pmod{p}$ (见练习题 3 (a))。因此至少 y 和 $-y$ 其中之一是一个平方数模 p 的余数。假设 y 和 $-y$ 都是一个平方数模 p 的余数，我们就说 $y \equiv a^2$，$-y \equiv b^2$。那么 $-1 \equiv (a/b)^2$ (模 p 分数的计算见 3.3 节)，说明 -1 就是一个平方数模 p 的余数。当 $p \equiv 3 \pmod{4}$ 时这是不成立的 (见习题 15)。因此，y 和 $-y$ 恰有一个模 p 平方根。如果 y 有一个模 p 平方根，那么 $y \equiv x^2$，y 模 p 的两个平方根就是 $\pm x$。如果 $-y$ 有一个模 p 平方根，那么 $x^2 \equiv -y$。证毕。 □

例如: 让我们看一下 5mod 11 的平方根。由于 $(p+1)/4 = 3$，我们算出 $x \equiv 5^3 \equiv 4 \pmod{11}$。又由于 $4^2 \equiv 5 \pmod{11}$，那么 5 mod 11 的平方根就等于 ± 4。

现在让我们试着找一下 2 mod 11 的平方根，由于 $(p+1)/4 = 3$，我们算出 $2^3 \equiv 8 \pmod{11}$，但是 $8^2 \equiv 9 \equiv -2 \pmod{11}$，所以可以得到一个根 -2 而不是 2。这是因为 2 没有模 11 平方根。

■

现在我们考虑一下对合数取模的平方根的情况。注意

$$x^2 \equiv 71 \pmod{77}$$

意味着

$$x^2 \equiv 71 \equiv 1 \pmod 7 \text{ 和 } x^2 \equiv 71 \equiv 5 \pmod{11}。$$

因此，

$$x \equiv \pm 1 \pmod 7 \text{ 和 } x \equiv \pm 4 \pmod{11}。$$

中国剩余定理告诉我们，mod 7 同余和 mod 11 同余可以结合成 mod 77 同余。例如：如果 $x \equiv 1 \pmod 7$ 且 $x \equiv 4 \pmod{11}$，那么 $x \equiv 15 \pmod{77}$。利用这种方法，可以通过 4 种途径来得到解

$$x \equiv \pm 15, \pm 29 \pmod{77}。$$

现在让我们回到刚才的讨论。假设 $n = pq$ 是两个素数的乘积，并且我们知道了 $x^2 \equiv y \pmod n$ 的 4 个解 $x \equiv \pm a, \pm b$。通过前面的解释，我们知道了 $a \equiv b \pmod p$ 和 $a \equiv -b \pmod q$（或者是 q 与 p 对等变换）。因此，$p \mid (a-b)$ 而不是 $q \nmid (a-b)$。这就意味着 $\gcd(a-b, n) = p$，所以我们已经发现了一个 n 的非平凡因子（基本原理见 6.3 节）。

例如，在先前的例子当中，我们知道 $15^2 \equiv 29^2 \equiv 71 \pmod{77}$。因此，$\gcd(15-29, 77) = 7$，给出了 77 的一个非平凡因子。

另外一个关于计算模 n 平方根的例子见 11.1 节。

注意到上面所使用的方法速度都很快，因数分解 n 例外。尤其是中国剩余定理计算很快，当然 gcd 的计算也一样。使用有效的平方技术，可以加快模 p 和模 q 平方根的计算。而这正是模的幂计算所需要的。所以，我们可以总结陈述如下原理：

假设 $n = pq$ 是两个同余于 3 模 4 的素数乘积，并且假设 y 与 n 是互素的，并且 y 有一个模 n 平方根，那么求出 $x^2 \equiv y \pmod n$ 的 4 个解 $x \equiv \pm a, \pm b$ 与对 n 的因数分解在计算方面是等同的。

换句话说，如果我们能够找到解，那么我们就能很容易地对 n 进行因数分解；反过来，如果我们能够对 n 进行因数分解，我们也能够很容易地找到这 4 个解。

3.10 有限域

备注：*这一节的内容比本章中其余内容都更为先进。之所以被包括进来，是因为在密码学中要经常用到它。特别是，有限域的概念在本书中 4 个地方出现过。有限域 $GF(2^8)$ 使用于 Rijndael（第 5 章）。有限域对 2.11 节中提到的一些现象给予了解释。最后，有限域还在 15.4 节和纠错码（第 16 章）中分别得以使用。*

在整本书中，我们多次提到了模 p 整数，这里 p 是素数。它可以进行加、减、乘运算，但是模 p 运算与对模任意整数 n 的运算的差别就在于模 p 运算可用任意非零的数去除。例如，如果需要解 $3x \equiv 1 \pmod 5$，可以用 3 去除，得 $x \equiv 2 \pmod 5$。与此相比，如果我们想解 $3x \equiv 1 \pmod 6$，因为不能通过 $3 \pmod 6$ 进行除，所以没有解。不严格地说，能与非零元素进行加、减、乘、除操作的集合就称为一个域。我们也要求掌握在域上的结合律、交换律和分配律。

例：域的基本例子就是实数、复数、有理数和模某个素数的整数集。所有整数的集合并不是一个域，因为有时不能进行除运算以获得该集合上的某个解（如：4/3 就不是一个整数）。■

例：有一个带有 4 个元素的域，如下

$$GF(4)=\{0,1,\omega,\omega^2\},$$

使用下面的定律：

1. 对所有 $x, 0+x=x$。
2. 对所有 $x, x+x=0$。
3. 对所有 $x, 1\cdot x=x$。
4. $\omega+1=\omega^2$。
5. 加法和减法满足交换律、结合律，并且对于所有 x，y，z，满足分配律 $x(y+z)=xy+xz$。

由于

$$\omega^3=\omega\cdot\omega^2=\omega\cdot(1+\omega)=\omega+\omega^2=\omega+(1+\omega)=1,$$

我们看到 ω^2 是 ω 的乘法逆元素。因此，每一个 GF(4) 的非零元素都有一个乘法逆元素，并且 GF(4) 是一个带有 4 个元素的域。■

总之，**域（field）**就是一个包含元素 0 和 1（$1\neq 0$）的集合，并且满足下列要求：

1. 在加和乘运算中，它必须满足先前所列的（1）、（3）、（5）定律；
2. 每一元素都有一个加法逆元素（对于每一个 x，这就意味着存在 $-x$ 使得 $x+(-x)=0$）；
3. 每一非零元素都有一个乘法逆元素。

一个域在减运算下是封闭的。为了计算 $x-y$，只需要简单计算 $x+(-y)$。

带有实元素的 2×2 矩阵集合并不是一个域，这有以下两个原因。首先，乘是不可交换的，其次，存在没有逆矩阵的非零矩阵（因此不能用它们进行除运算）。这种非负实数的集合就不是一个域。我们可以进行加、乘、除运算，但是有时进行减运算的时候，结果并不在该集合中。

对于一个素数的任意幂次方 p^n，恰存在一个带有 p^n 个元素的有限域，并且只有这一个有限域。我们很快将看到如何构建它们，但首先指出，如果 $n>1$，那么模 p^n 整数集不构成一个域。同余式 $px\equiv 1 \pmod{p^n}$ 没有解，所以即使 $p\not\equiv 0 \pmod{p^n}$，也不能通过除来得到解。因此我们需要更为复杂的方法来产生带有 p^n 个元素的域。

带有 p^n 个元素的域称为 $GF(p^n)$。"GF" 代表"伽罗瓦域（Galois field）"，是根据法国数学家埃瓦茨·伽罗瓦（Evariste Galois）（1811～1832 年）来命名的，他做了有关域的早期工作。

继续举例加以说明。有产生域 $GF(4)$ 的另外一种方法。设 $\mathbf{Z}_2[X]$ 是多项式集合，并且其系数是对 2 取模的整数。例如，$1+X^3+X^6$ 和 X 就在这个集合中。同时，常量多项式 0 和 1 也在 $\mathbf{Z}_2[X]$ 中。在该集合中，只要处理有关对 2 取模的系数，我们就可以进行加、减、乘运算。例如：

$$(X^3+X+1)(X+1)=X^4+X^3+X^2+1$$

因为 $2X$ 项对 2 取模而消失了。我们所要的最重要属性就在于，我们能够像使用整数进行除一

样使用余数。例如，假设我们用 X^2+X+1 来除 X^4+X^3+1。可以通过长除法来进行，如下：

$$\begin{array}{r} X^2+1 \\ X^2+X+1 \overline{\smash{)}\, X^4+X^3+1} \\ \underline{X^4+X^3+X^2} \\ X^2+1 \\ \underline{X^2+X+1} \\ X \end{array}$$

换句话说，我们所做的就是用 X^2+X+1 来进行除运算，并且获得 X^2 作为商的第一项。然后对 X^2+X+1 乘 X^2 得到 $X^4+X^3+X^2$，减去 X^4+X^3+1，剩下 X^2+1。再用 X^2+1 来除 X^2+X+1 以得到商的第二项即 1。对 X^2+X+1 乘 1 并且减去 X^2+1 剩下余数 X。既然多项式 X 的阶小于 X^2+X+1 的阶，就可以停止了。商是 X^2+1 并且余数是 X：

$$X^4+X^3+1=(X^2+1)(X^2+X+1)+X,$$

可以写成：

$$X^4+X^3+1\equiv X(\bmod X^2+X+1)。$$

无论什么时候使用 X^2+X+1 去除，都得到一个要么是 0 要么是阶数至少为 1 的多项式（如果阶数为 2 或更高的话，可以继续进行除）。因此，我们定义 $\mathbf{Z}_2[X](\bmod X^2+X+1)$ 是多项式阶数至少为 1 的集合

$$\{0,1,X,X+1\}$$

因为当用 X^2+X+1 进行除时，这些就是所得的余数。加、减、乘都是用模 X^2+X+1 进行的。这完全类似于模 n 整数[1]的情况。在现在的情况下，如果 $f(X)$ 和 $g(X)$ 被 X^2+X+1 除且有相同的余数，我们就说多项式 $f(X)$ 同余于多项式 $g(X)$ 对 X^2+X+1 取模，记作 $f(X)\equiv g(X)(\bmod X^2+X+1)$。另外一种说法就是 $f(X)-g(X)$ 是 X^2+X+1 的倍数，这意味着存在一个多项式 $h(X)$，使得 $f(X)-g(X)=(X2+X+1)h(X)$。

现在让我们在 $\mathbf{Z}_2[X](\bmod X^2+X+1)$ 中进行乘运算。例如：

$$X\cdot X=X^2\equiv X+1(\bmod X^2+X+1)。$$

（也许右边应该是 $-X-1$，但考虑到正在用模 2 系数处理，所以 $+1$ 和 -1 是相同的。）另外一个例子，

$$X^3\equiv X\cdot X^2\equiv X\cdot(X+1)\equiv X^2+X\equiv 1(\bmod X^2+X+1)。$$

很容易就可以看出，我们正是使用前面的集合 GF（4），用 X 来代替 ω。■

用 $\mathbf{Z}_2[X]$ 对一个多项式取模可以用来产生有限域，但是并不能对任意一个多项式取模。这个多项式必须是不可约的，即它不可以再分成更小的模 2 阶数的多项式。举个例子来说，对于 X^2+1，当处理实数时它是不可约的，但是当系数是模 2 的，它就不是不可约的，因为在模 2 的情况下 $X^2+1=(X+1)(X+1)$。然而，X^2+X+1 是不可约的：假如将它模 2 分解为更低阶的多项式。惟一可能的模 2 因子就是 X 和 $X+1$，并且即使是对 2 取模，X^2+X+1 也不是这些元素的倍数。

这有一个关于带有 p^n 个元素的有限域的通用构建过程，在这里 p 是一个素数并且 $n\geq 1$。令 $\mathbf{Z}p$ 来表示模 p 整数。

1　译者注：模 n 整数指的是整数对 n 取模后的余数。后文中类似的模 p 随机数、模 p 平方数指的是随机数、平方数对 p 求模后的余数。

1. $\mathbf{Z}_p[X]$ 是模 p 系数的多项式的集合。

2. 选取 $P(X)$ 为阶数是 n 的不可约的模 p 多项式。

3. 设 $GF(p^n)$ 为 $\mathbf{Z}_p[X]$ 模 $P(X)$ 的余数，则 $GF(p^n)$ 为具有 p^n 个元素的域。

$GF(p^n)$ 有 p^n 个元素是很明显的。被 $P(X)$ 除过后可能的余数多项式为 $a_0 + a_1X + \cdots + a_{n-1}X^{n-1}$，这里的系数为模 p 整数的形式。每个系数有 p 种可能，因此有 p^n个可能的余数。

对于每个 n，n 阶模 p 多项式是不可约的，因此对于每个 $n \geqslant 1$，这种结构产生具有 p^n个元素的域。如果对两个不同的 n 阶多项式 $P_1(X)$ 和 $P_2(X)$ 做相同的构造，结果会如何呢？可以得到两个域，分别叫作 $GF(p^n)'$ 和 $GF(p^n)''$，其本质上为同一域（专业术语为这两个域同构），但由于模 $P_1(X)$ 乘法与模 $P_2(X)$ 乘法不一样，导致这种结果并不明显。

3.10.1 除法

在 $\mathbf{Z}_p[X]$ 里进行多项式加、减和乘是比较容易的，但除法有一点麻烦。举例来说，多项式 $X^8 + X^4 + X^3 + X + 1$ 在 $\mathbf{Z}_2[X]$ 域里是不可约的（虽然有更快的方法可表明它是不可约的，但常用的方法就是用 $\mathbf{Z}_2[X]$ 域里更低阶的所有多项式去除之）。考虑如下的域

$$GF(2^8) = \mathbf{Z}_2[X](\mathrm{mod}\ X^8 + X^4 + X^3 + X + 1),$$

因为 $X^7 + X^6 + X^3 + X + 1$ 不为 0，所以它有逆元素。用欧几里得的扩展算法来求其逆元素。首先，进行 $\gcd(X^7 + X^6 + X^3 + X + 1, X^8 + X^4 + X^3 + X + 1)$ 的最大公约数计算。此计算过程（余数→除数→被除数→忽略）与整数的计算是一样的。

$$X^8 + X^4 + X^3 + X + 1 = (X + 1)(X^7 + X^6 + X^3 + X + 1) + (X^6 + X^2 + X)$$

$$X^7 + X^6 + X^3 + X + 1 = (X + 1)(X^6 + X^2 + X) + 1$$

最后的余数是 1，这就告诉我们 $X^7 + X^6 + X^3 + X + 1$ 和 $X^8 + X^4 + X^3 + X + 1$ 的“最大公约数”是 1。当然，肯定是这样的，因为 $X^8 + X^4 + X^3 + X + 1$ 是不可再约的，所以因子只有 1 和它本身。

现在回过头来看，通过计算得知 1 是 $X^7 + X^4 + X^3 + X + 1$ 和 $X^8 + X^4 + X^3 + X + 1$ 的线性组合（或用扩展的欧几里得算法）。注意，在每一步里，我们取最终的无用余数，并且用被除数减去除数乘以商来替换它；因为进行的是模 2 运算，负号不会出现。

$$\begin{aligned}
1 &= (X^7 + X^6 + X^3 + X + 1) + (X + 1)(X^4 + X^2 + X) \\
&= (X^7 + X^4 + X^3 + X + 1) \\
&\quad + (X + 1)\big((X^8 + X^4 + X^3 + X + 1) + (X + 1)(X^7 + X^6 + X^3 + X + 1)\big) \\
&= (1 + (X + 1)^2)(X^7 + X^6 + X^3 + X + 1) + (X + 1)(X^8 + X^4 + X^3 + X + 1) \\
&= (X^2)(X^7 + X^6 + X^3 + X + 1) + (X + 1)(X^8 + X^4 + X^3 + X + 1),
\end{aligned}$$

因此，

$$1 = (X^2)(X^7 + X^6 + X^3 + X + 1) + (X + 1)(X^8 + X^4 + X^3 + X + 1),$$

减去模 $X^8 + X^4 + X^3 + X + 1$，得到

$$(X^2)(X^7 + X^8 + X^3 + X + 1) \equiv 1(\mathrm{mod}\ X^8 + X^4 + X^3 + X + 1),$$

这意味着 X^2是 $X^7 + X^6 + X^3 + X + 1$ 的乘法逆元素。当需要用 $X^7 + X^6 + X^3 + X + 1$ 作除数时，就可以用乘以 X^2 来代替。这与普通的模 p 整数运算是类似的。

GF（2^8）

在本书的后面，我们将讨论 Rijndael，它使用 $GF(2^8)$（参见第 5 章），因此必须仔细阅读这一部分。我们将使用对不可约多项式 $X^8+X^4+X^3+X+1$ 取模的情况，因为这正是 Rijndael 使用的。但是，也有其他不可约的 8 阶多项式，它们中的任意一个都会有相似的运算。每个元素都可用惟一的多项式来表示：

$$b_7X^7+b_6X^6+b_5X^5+b_4X^4+b_3X^3+b_2X^2+b_1X+b+0,$$

这里 b_i 为 0 或 1，8 比特的 $b_7b_6b_5b_4b_3b_2b_1b_0$ 代表一个字节，所以可以用 $GF(2^8)$ 的元素来代表 8 比特字节。例如，多项式 $X^7+X^6+X^3+X+1$ 变成 11001011。加法就是对应位的异或 XOR 运算：

$$\begin{aligned}&(X^7+X^6+X^3+X+1)+(X^4+X^3+1)\\&\to 11001011 \oplus 00011001 = 11010010\\&\to X^7+X^6+X^4+X。\end{aligned}$$

乘法是比较敏感的，相对来说不容易解释。这是因为工作在对多项式 $X^8+X^4+X^3+X+1$ 取模的模式下，它可以用 9 比特数 100011011 来表示。首先，用 X 来乘 $X^7+X^6+X^3+X+1$。用多项式，我们计算如下：

$$\begin{aligned}(X^7+X^6+X^3+X+1)(X) &= X^8+X^7+X^4+X^2+X\\&=(X^7+X^3+X^2+1)+(X^8+X^4+X^3+X+1)\\&\equiv X^7+X^3+X^2+1 \pmod{X^8+X^4+X^3+X+1}。\end{aligned}$$

同样的比特操作如下：

$$\begin{aligned}11001011 \to &\quad 110010110 \quad (\text{左移并附加一个 } 0)\\ \to &\quad 110010110 \oplus 100011011 \quad (\text{减去 } X^8+X^4+X^3+X+1)\\ = &\quad 010001101,\end{aligned}$$

这与前述的答案一致。通常，我们可以用如下的算法乘 X：

1. 左移并在最后一位附加一个 0。
2. 如果第一位是 0，则停止。
3. 如果第一位是 1，与 100011011 进行异或操作。

第二步停止是因为如果第一位为 0，乘以 X 后多项式的阶数小于 8，所以没有必要简化。为了乘更高阶的 X，需要数次乘以 X，例如乘以 X^3 可以左移三次，最多进行三次异或操作。任意多项式的乘法，可以通过乘以此多项式中出现的各个 X 的幂，然后将结果相加（异或）得到。

总之，我们可以看到在 $GF(2^8)$ 域中的加与乘操作可以有效地得到执行。同样的考虑可以应用于其他有限域。

整数模一个素数与多项式模一个不可约的多项式这两种情况有很多类似的地方，总结如下：

$$\begin{aligned}\text{整数} &\quad\leftrightarrow\quad \mathbf{Z}_p[X]\\ \text{素数 } q &\quad\leftrightarrow\quad n \text{ 阶不可约 } P(X)\\ \mathbf{Z}_q &\quad\leftrightarrow\quad \mathbf{Z}_p[X] \pmod{P(X)}\\ \text{具有 } q \text{ 个元素的域} &\quad\leftrightarrow\quad \text{具有 } p^n \text{ 个元素的域}\end{aligned}$$

设 $GF(p^n)^*$ 表示 $GF(p^n)$ 中的非零元素。这个集合有 p^n-1 个元素，在乘法情况下是封

闭的，就像整数不同余于 0 mod p 在乘法下是封闭的一样。从这可以看出，存在一个生成多项式 $g(X)$ ，使得在 $GF(p^n)^*$ 中的任一元素都可以用 $g(X)$ 的幂来表示。这也意味着满足 $g(X)^k \equiv 1$ 的最小指数 k 为 $p^n - 1$ 。这与素数的本原根类似。有 $\phi(p^n - 1)$ 这样的生成多项式，这里 ϕ 是欧拉函数，当 $p = 2$ 和 $2^n - 1$ 是素数时会产生有趣的结果，在这种情况下，$GF(2^n)$ 中的任何非零多项式 $f(X) \neq 1$ 都是一个生成多项式。[注意，对于知道群论的人，在这种情况下，集合 $GF(2^n)^*$ 为素有序群，因此除了其自身，每个元素都是一个产生器。]

至于模一个素数的**离散对数问题（discrete log problem）**，我们将在第 7 章中讨论，它与有限域有些类似；即，给定 $h(x)$ ，找出一个整数 k，使在 $GF(p^n)$ 中满足 $h(X) = g(X)^k$ ，这样的 k 在大多数情形下较难求得。

3.10.2 LFSR 序列

现在我们来解释在 2.11 节中所提到的 LFSR 现象。

以递归的形式开始，比如

$$x_{n+4} \equiv x_n + x_{n+1} \pmod 2,$$

如果初始值是

$$x_0 = 1, x_1 = 1, x_2 = 0, x_3 = 1$$

序列是

1 1 0 1 0 1 1 1 1 0 0 0 1 0 0 1 1 0 1 ...

使用的递归表示成多项式为

$$X^4 + X + 1$$

在这种情况下是模 2 不可约的，因此 $\mathbf{Z}_2[X] \pmod{X^4 + X + 1}$ 是有 16 个元素的域 $GF(16)$ ，用至多 3 阶多项式表示，对初始值 1 1 0 1，结合 $GF(16)$ 中的元素 $1 + X + X^3$，与 X 相乘结果模 $X^4 + X + 1$：

$$X(1 + X + X^3) = X + X^2 + X^4 \equiv X + X^2 + (X + 1) \equiv 1 + X^2 \pmod{X^4 + X + 1}$$

多项式 $1 + X^2$ 对应的数字是 1 0 1 0 ，就是 x_1, x_2, x_3, x_4 。现在 $1 + X + X^3$ 乘以 X^2 ，它与 $1 + X^2$ 乘以 X 是相同的，再将结果模 $X^4 + X + 1$ 得到：

$$X^2(1 + X + X^3) \equiv X(1 + X^2) \equiv X + X^3 \pmod{X^4 + X + 1},$$

它对应数字 0 1 0 1，即 x_2，x_3，x_4，x_5。按此法继续，有

$$X^k(1 + X + X^3) \equiv x_k + x_{k+1}X + x_{k+2}X^2 + x_{k+3}X^3 \pmod{X^4 + X + 1}$$

（由推论可以证明）。当 $x_k = x_0, x_{k+1} = x_1, x_{k+2} = x_2, x_{k+3} = x_3$ 时，序列开始重复，这表明

$$X^k(1 + X + X^3) \equiv 1 + X + X^3 \pmod{X^4 + X + 1},$$

用 $1 + X + X^3$ 去除，得到

$$X^k \equiv 1 \pmod{X^4 + X + 1}。$$

$k = 15$ 是最小的正数，于是经过 15 项后序列重复，从前面的序列元素表也能看出。对于 $k < 15, X^k \neq 1$ 表示 X 的幂是完全不同的，这样 X 的幂给出了所有 15 个在 $GF(16)$ 中的非零元素。因此 X 是一个生成多项式。

一般情况是类似的。为了简单起见，假定如下形式

$$x_{n+m} \equiv c_0x_n + c_1x_{n+1} + \cdots + c_{m-1}x_{n+m-1} \pmod 2$$

是一个递归关系式，其表示的多项式是

$$P(X) = X^m + c_{m-1}X^{m-1} + c_{m-2}X^{m-2} + \cdots c_0$$

是模 2 不可约的，有 $\mathbf{Z}_2[X] \pmod{P(X)}$ 在域 $GF(2^m)$ 中。对任意初始值，除了所有 0 组成的序列，都应有周期 k，k 是满足 $X^k \equiv 1 \pmod{P(X)}$ 的最小正整数。

正如前面讨论的那样，当 $2^m - 1$ 是素数时，所有的多项式（除 0 和 1 外）对 $GF(2^m)$ 都是生成多项式。特别地，X 是生成多项式并且 $k = 2^m - 1$ 是递归的周期。

3.11 习　　题

1. (a) 求整数 x 和 y，使得 $17x + 101y = 1$。

(b) 求 $17^{-1} \pmod{101}$。

2. (a) 解 $7d \equiv 1 \pmod{30}$。

(b) 假如有一组信息表示为 $m \pmod{31}$，把 m 加密为 $m^7 \pmod{31}$，如何解密？(提示：将密文提高到模 31 的幂，再运用费尔马定理。)

3. (a) 已知 p 是一个素数，假如 a 和 b 都是整数，且 $ab \equiv 0 \pmod p$。试证：$a \equiv 0$ 或 $b \equiv 0 \pmod p$。

(b) 试证：如果 a，b，n 是整数、且 $n \mid ab, \gcd(a,n) = 1$，则 $n \mid b$。

4. 已知 p 是一个素数 $(p \geqslant 3)$，求证：$X^2 \equiv 1 \pmod p$ 的惟一解是 $x \equiv \pm 1 \pmod p$。(提示：运用练习题 3 (a) 转换为 $(x+1)(x-1)$。)

5. 假设 $x \equiv 2 \pmod 7$ 且 $x \equiv 3 \pmod{10}$，求与模 70 同余的 x 是多少？

6. 一群人排队列，如果 3 人一行，则余 1 人，如果 4 人一行，则余 2 人，如果 5 人一行，则余 3 人。问：至少有多少人？下一个最小的数是多少？(提示：用中国剩余定理解释这个问题。)

7. 设 p 是一个素数，试证：对于所有的 $a, a^p \equiv a \pmod p$。

8. (a) 设 $p = 7$，13 或 19，试证：对于所有的 a $(p \nmid a)$，有 $a^{1728} \equiv 1 \pmod p$。

(b) 设 $p = 7$，13 或 19，试证：对于所有的 a，有 $a^{1729} \equiv a \pmod p$。(提示：单独考虑 $p \mid a$ 的情况。)

(c) 试证：对于所有的 a，有 $a^{1729} \equiv a \pmod{1729}$。对于所有的 a，如果合数 n 满足 $a^n \equiv a \pmod n$，则称 n 为 Carmichael 数，这种数相对稀少 (561 是另外一例)，但是数量是无限多的。

9. 设整数 a 和 $n(n > 1)$，且 $\gcd(a,n) = 1$，满足条件 $a^r \equiv 1 \pmod n$ 的最小正整数 r 称为 $a \mod n$ 的**序号 (order)**，记作 $r = \mathrm{ord}_n(a)$。

(a) 证明：$r \leqslant \phi(n)$。

(b) 证明：如果 $m = rk$ 是 r 的倍数，那么 $a^m \equiv 1 \pmod n$。

(c) 设 $a^t \equiv 1 \pmod n$，$t = qr + s$ 且 $0 \leqslant s < r$ (这是带有余数的除法)。证明：$a^s \equiv 1 \pmod n$。

(d) 已知 r 的定义，且 $0 \leqslant s < r$，证明：$s = 0$，且 $r \mid t$，由本题的结论和 (b) 中的结论我们可以得出：当且仅当 $\mathrm{ord}_n(a) \mid t$ 时，$a^t \equiv 1 \pmod n$。

(e) 证明：$\mathrm{ord}_n(a) \mid \phi(n)$。

10. 这个练习将举例说明，已知 $p-1$ 的因数分解，如何利用练习 9 的结论证明一个数是模素数 p 本原根。特别地，还将说明 7 是模 601 本原根。注意：$600 = 2^3 \cdot 3 \cdot 5^2$。

(a) 试证明：如果 600 能够被一个整数 $r < 600$ 整除，那么至少 300，200，120 其中之一能被 r 整除（这 3 个数分别是 600/2，600/3，600/5）。

(b) 试证明：如果 $\mathrm{ord}_{601}(7) < 600$，那么它能够整除 300，200，120 的其中一个。

(c) 下列计算显示：

$$7^{300} \equiv 600,\ 7^{200} \equiv 576,\ 7^{120} \equiv 423 \pmod{601},$$

为什么 ord_{601}（7）不能够整除 300，200，120 的其中之一？

(d) 试证明：7 是模 601 本原根。

(e) 一般而言，如果 p 是素数，且 $p-1 = q_1^{a_1} \cdots q_s^{a_s}$ 是它的因数分解，描述一个过程来判断一个数 g 是否模 p 本原根。（因此，如果需要求模 601 本原根，我们只需要用这个过程来依次判断 $g = 2, 3, 5, 6, \ldots$，直至找到本原根。）

11. 如果我们要求指数 k，使得 $3^k \equiv 2 \pmod{65537}$。

(a) 注意到 $2^{32} = 1 \pmod{65537}$，但是 $2^{16} \not\equiv 1 \pmod{65537}$，由此可以推出 3 是模 65537 本原根，它表明 $3^n \equiv 1 \pmod{65537}$ 当且仅当 $65536 \mid n$。利用该结论证明，$2048 \mid k$ 但 k 不能被 4096 整除。（提示：提高 $3^k \equiv 2$ 两边到第 16 和 32 个幂次方。）

(b) 使用（a）中的结果推导出，需要考虑的 k 值只有 16 种可能的取值。利用这一点来求 k。这也说明了，如果 $p-1$ 有特殊的结构，比如 2 的幂，那么它可以用来避免穷尽搜索。因此，这些素数就成为加密的弱点。此问题的进一步解释见练习 7.5。

12. (a) 设 $x = b_1 b_2 \ldots b_w$ 是二进制整数的形式（例如，当 $x = 1011$ 时，有 $b_1 = 1, b_2 = 0, b_3 = 1, b_4 = 1$），设 y 和 n 是整数，进行如下计算：

(1) 初始条件：$k = 1$，$s_0 = 1$；

(2) 如果 $b_k = 1$，设 $r_k \equiv s_k y \pmod{n}$；如果 $b_k = 0$，设 $r_k \equiv s_k$；

(3) 设 $s_{k+1} \equiv r_k^2 \pmod{n}$；

(4) 如果 $k = w$，则结束循环；如果 $k < w$，则 $k = k+1$，然后继续执行（2）。

试证明：$r_w \equiv y^x \pmod{n}$。

(b) 设 x，y，n 都是正整数，试证明：下列过程能够计算 $y^x \pmod{n}$。

(1) 初始条件：$a = x, b = 1, c = y$；

(2) 如果 a 是偶数，设 $a = a/2$，并设 $b = b$，$c = c^2 \pmod{n}$；

(3) 如果 a 是奇数，设 $a = a - 1$，并设 $b \equiv bc \pmod{n}$，$c = c$；

(4) 如果 $a \neq 0$，执行第二步；

(5) 输出 b。

（注：这种算法与（a）十分相似，但它使用的是 x 反序的二进制位。）

13. 这里解释怎样利用中国剩余定理构建 x。设整数 $m_1, \ldots, m_k$，且 $\gcd(m_i, m_j) = 1$ $(i \neq j)$。设 $a_1, \ldots, a_k$ 是整数，执行下列步骤：

(1) 当 $i = 1, \ldots, k$ 时，设 $z_i = m_1 \cdots m_{i-1} m_{i+1} \cdots m_k$；

(2) 当 $i = 1, \ldots, k$ 时，设 $y_i \equiv z_i^{-1} \pmod{m_i}$；

(3) 设 $x = a_1 y_1 z_1 + \cdots + a_k y_k z_k$；

试证：对于所有的 $i, x \equiv a_i \pmod{m_i}$。

14. (a) 求 $x^2 \equiv 133 \pmod{143}$ 的所有 4 个解。（注意：$143 = 11 \cdot 13$。）

(b) 求 $x^2 \equiv 77 \pmod{143}$ 的所有解。（本题只有两个解，因为 $\gcd(77,143) \neq 1$。）

15. 设 $p \equiv 3 \pmod 4$ 是素数，试证：$x^2 \equiv -1 \pmod p$ 无解。（提示：假设 x 存在，提高两边到幂 $(p-1)/2$，再应用费尔马定理。）

16. 艾丽斯设计了下面这个加密体制（这个体制基于 Rabin），她选择了两个不同的素数 p，q（最好 p 和 q 对 3 模 4 是同余的），当然，p，q 是不公开的。她只让 $n = pq$ 公开，当鲍勃想要发送一段信息 m 给艾丽斯时，他计算 $x \equiv m^2 \pmod n$，并且将 x 发送给艾丽斯，艾丽斯使用一个解密机器做如下工作：当接收到数字 m 时，由于它知道 p，q 的值，于是计算 $x \pmod n$ 的平方根。并且，通常情况下，可以得到不止一个平方根。平方根的选择是随机的，选择之后，发送给艾丽斯。当艾丽斯接收到鲍勃发送的信息 x，将它输入到解密机器中进行处理，如果输出是一个有意义的信息，那么艾丽斯就把它当作是正确的信息，如果输出不是一个有意义的信息，那么艾丽斯再次把它放入解密机器中进行处理，直到得到一个有意义的信息。

(a) 为什么艾丽斯期望在最快的时间内得到有意义的信息？

(b) 如果奥斯卡截获了 x（他已经知道了 n），为什么对他而言要得到信息 m 非常困难？

(c) 如果伊芙闯入到艾丽斯的办公室，在那里，他可以通过尝试一些可选择密文来攻击艾丽斯的解密机器，问：他如何确定 n 的因数分解？

17. 本题显示了欧几里得算法计算 gcd 的过程。设 a, b, q_i, r_i 和 3.1 节中定义的一样，

(a) 设 d 是 a，b 的公约数，试证：$d \mid r_1$，并且利用这个结果证明 $d \mid r_2$。

(b) 设 d 是 a，b 的公约数，利用归纳法证明：对于所有的 i，都有 $d \mid r_i$，特别地，$d \mid r_k$ 是最后一个非零余数。

(c) 利用归纳法证明：对于 $1 \leq i \leq k$，有 $r_k \mid r_i$。

(d) 利用 $r_k \mid r_1$ 和 $r_k \mid r_2$，证明：$r_k \mid b, r_k \mid a$，因此，r_k 是 a, b 的公约数。

(e) 利用 (b) 中的结论，证明：对于所有的公约数 d，都有 $r_k \geq d$，因此，r_k 是最大公约数。

18. (a) 试证明：$\mathbf{Z}_2[X]$ 中至多为 2 阶的惟一不可约的多项式是 $X, X+1, X^2+X+1$。

(b) 证明：在 $\mathbf{Z}_2[X]$ 中，X^4+X+1 是不可约的。（提示：如果它是因子，则一定有至少一个阶数至多是 2 的因子。）

(c) 证明：$X^4 \equiv X+1, X^8 \equiv X^2+1$，且 $X^{16} \equiv X \pmod{X^4+X+1}$。

(d) 证明：$X^{15} \equiv 1 \pmod{X^4+X+1}$。

19. (a) 试证明：X^2+1 在 $\mathbf{Z}_3[X]$ 中是不可约的。

(b) 求在 $\mathbf{Z}_3[X] \pmod{X^2+1}$ 中 $1+2X$ 的乘法逆元素。

3.12 上　机　题

1. 计算 gcd（8765，23485）。

2. (a) 求整数 x 和 y，使得 $65537x + 3511y = 1$。

(b) 求整数 x 和 y，使得 $65537x + 3511y = 17$。

3. 找出 $3^{1234567}$ 的最后 5 位数字。（注意：不要用计算机计算 $3^{1234567}$，结果太大了！）

4. 解 $314x \equiv 271 \pmod{11111}$。

5. 求 $216x \equiv 66 \pmod{606}$ 的所有解。

6. 求整数 x，使得当它被 101 整除的时候，余数为 17，当它被 201 整除的时候，余数为 18，当它被 301 整除的时候，余数为 19。

7. 设 $n = 391 = 17 \cdot 23$，证明 $2^{n-1} \not\equiv 1 \pmod{n}$，找出指数 $j>0$，使之满足 $2^j \equiv 1 \pmod{n}$。

8. 设 $n = 84047 \cdot 65497$，求 x 和 y，使得 $x^2 \equiv y^2 \pmod{n}$，但是 $x \not\equiv \pm y \pmod{n}$。

9. 证明 3 是 65537 的本原根。（提示：用练习 10 的方法。）

10. 设 $M = \begin{pmatrix} 1 & 2 & 4 \\ 1 & 5 & 25 \\ 1 & 14 & 196 \end{pmatrix}$

(a) 求 $M \pmod{101}$ 的逆矩阵。

(b) 素数 p 等于多少时，M 没有模 p 逆矩阵？

11. 求 26055 模素数 34807 的平方根。

12. 求 1522756 (mod 2325781) 的所有平方根。

13. 用 3.9 节的方法，求 48382 模素数 83987 的平方根，并验证它是否正确。你用此方法求出的平方根是多少？

第4章 数据加密标准

4.1 概 述

1973 年，国家标准局（NBS）即后来的国家标准和技术委员会（NIST）发布了一个公开请求，寻找一个加密算法使之成为国家标准。IBM 在 1974 年提出了一个算法叫 LUCIFER，NBS 后来将该算法提交给国家安全代理机构，它们重新审阅并对算法做了一些修改，提出了一个版本，就是最初的数据加密标准（DES）算法。1975 年，NBS 公开了 DES，即可以自由地使用它，并于 1977 年将它作为政府数据加密标准。

DES 广泛地使用在电子商务中，比如银行业，如果两个银行要交换数据，他们首先要使用公开密钥方法如 RSA 来传送 DES 密钥，然后使用 DES 来传送数据，它具有快速和安全的优点。

自 1975 年以来，围绕 DES 有相当激烈的争论，一些人认为这个密钥太短，更多的人担心受 NSA 的牵连。例如，他们担心有一个“陷门”——换句话就是担心会有一个秘密的弱点，从这里来攻击系统。因此就有人建议 NSA 修改设计，以避免 IBM 在 LUCIFER 中插人一个陷门的可能性，无论如何，很多年以来这个设计一直是一个秘密。

1990 年，*Eli Biham* 和 *Adi Shamir* 提出了如何用微分密码分析法来破译 DES。DES 算法有 16 个循环；微分分析法在算法至多有 15 个循环的情况下，比起穷尽搜索所有可能的密钥有更高的效率。几年后，IBM 公开了一些设计标准，它表明实际上他们已建立了阻止微分密码分析的体制，这多少解除了关于算法的神秘面纱。

DES 持续了相当长的时间才过时，人们夜以继日、不惜一切代价地研究（见 4.6 节），现在终于破译了 DES。因此，NIST 在 2000 年用新的体制取代了它，但是，DES 是很值得研究的，因为它代表了一类流行的算法，且是历史上使用最频繁的密码学算法之一。

DES 属于分组密码，也就是说它将明文每 64 比特[1] 分成一组，并独立地加密每一组。自 Horst Feistel 出现以来，他是 IBM 中提出 LUCIFER 算法的成员之一，实际中怎样实施加密的技工们就常称之为 **Feistel 体制（Feistel system）**。在后面的章节中，我们将给出具有许多这类体制特点的一个简单算法，但是它很小，可用来作为例子。在 4.3 节中，我们将讲到怎样用微分密码分析法来破解这个简单系统，在 4.4 节中给出了 DES 算法，4.5 节中讲述了它

---

1 译者注：为简单起见，本章中所涉及的位均是指二进制的比特位

的实施，最后，在4.6节中将描述在破译DES方面的最新进展。

对分组密码的详细描述见［Schneier］。

4.2 一个简单的类DES算法

DES算法作为例子来讲解的话相当大，很不现实，所以本节中我们提出了一个具有很多DES特点的算法，而它又小得多，类似于DES，也是分组密码。由于分组密码是各自独立加密的，所以假设所讨论的全部信息仅是某一个分组的组成。

信息有12位并表示为L_0R_0的形式，这里L_0是指前6位，而R_0是指后6位。密钥K有9位，算法的第i个循环利用从密钥K中取出的一个8位密钥K_i转换传送的输入$L_{i-1}R_{i-1}$，得到输出L_iR_i。

算法加密过程的主要部分就是函数$f(R_{i-1}, K_i)$，它输入6位的R_{i-1}和8位的K_i，产生一个6位的输出，后而将讲述这个函数。

第i个循环的输出定义如下：

$$L_i = R_{i-1} \text{ 和 } R_i = L_{i-1} \oplus f(R_{i-1}, K_i),$$

这儿$\oplus$表示异或，即逐位模2相加，详见图4.1。

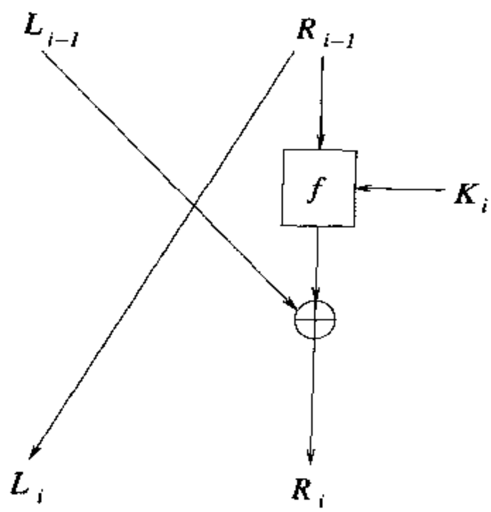


图4.1 Feistel体制的一次循环

执行某个循环叫n的操作，它产生的密文是L_nR_n。

怎样来解密呢？从L_nR_n开始，左右两边交换以得到R_nL_n（注：这个交换是用DES加密算法实现的，在解密DES时，不需要它）。现在使用同前一样的过程，只是使用的密钥K_i是前面密钥序列的反转，即按$K_n, \ldots, K_1$的顺序。下面来看一下它的工作情况，首先从R_nL_n开始，得到输出

$$[L_n]\,[R_n \oplus f(L_n, K_n)],$$

从加密过程中我们可以知道$L_n = R_{n-1}$和$R_n = L_{n-1} \oplus f(R_{n-1}, K_n)$。因此，

$$[L_n]\,[R_n \oplus f(L_n, K_n)] = [R_{n-1}]\,[L_{n-1} \oplus f(R_{n-1}, K_n) \oplus f(L_n, K_n)]$$

$$= [R_{n-1}][L_{n-1}]$$

最后一个等式又一次使用了 $L_n = R_{n-1}$，这样 $f(R_{n-1}, K_n) \oplus f(L_n, K_n)$ 等于 0。类似地，解密的第二步是发送 $R_{n-1}L_{n-1}$ 转换成 $R_{n-2}L_{n-2}$，如此继续，我们可看到，这个解密过程最终转换成 R_0L_0。从中间交换左右两边，可得到希望的明文 L_0R_0。

注意，解密的过程与加密过程基本相同，只需简单地交换左右两边，并使用反顺序的密钥 K_i，因此发送方和接收方均使用了同一组密钥，也就可以使用同一种机器（虽然接收方需要反转左和右输入）。

至此，我们还未说明函数 f 的任何事情，实际上任何函数都适用于上述过程，但某些函数比其他函数会产生好得多的安全性，后面将要讲到的用于 DES 的函数 f 都是非常类似的，它是由一些新组件构成的。

第一个函数是一个扩展器，它输入 6 比特输出 8 比特，图 4.2 显示了我们所使用的一个函数示例。

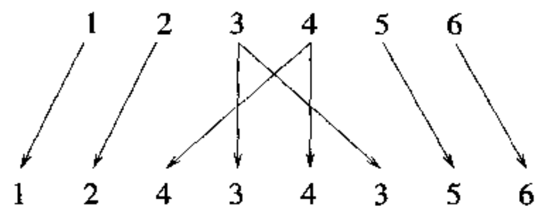


图 4.2　扩展函数

这意味着，第一位输入产生第一位输出，第三位输入产生第四位和第六位输出，等等。比如 011001 扩展输出成 01010101。

最主要的组件叫做 S-盒，我们使用了两个：

$$S_1 \quad \begin{bmatrix} 101 & 010 & 001 & 110 & 011 & 100 & 111 & 000 \\ 001 & 100 & 110 & 010 & 000 & 111 & 101 & 011 \end{bmatrix}$$

$$S_2 \quad \begin{bmatrix} 100 & 000 & 110 & 101 & 111 & 001 & 011 & 010 \\ 101 & 011 & 000 & 111 & 110 & 010 & 001 & 100 \end{bmatrix}$$

S-盒的输入有四位，第一位表示哪一行将被使用：0 代表第一行，1 代表第二行，其他的 3 位表示某一列的二进制数：000 表示第一列，001 代表第二列，. . . ，111 代表最后一列。S-盒的输出是指盒中某一个位置的 3 位的值，例如，对 S_1 盒输入 1010，意味着查看 S_1 的第二行、第三列，其输出就是 110。

密钥 K 由 9 位组成，第 i 个加密循环的密钥 K_i 从密钥 K 中第 i 个位置起的 8 位而得来。例如，$K = 010011001$，那么 $K_4 = 01100101$（经过 5 位后，到达 K 的最末，所以最后的两位从 K 的开始得到）。

现在我们来讲述函数 $f(R_{i-1}, K_i)$。输入 R_{i-1} 由 6 位组成，使用扩展函数将它扩展成 8 位，结果与 K_i 异或产生另一个 8 位的结果，这个结果的前 4 位发送给 S_1，后 4 位发送给 S_2，每一个 S 盒输出 3 位，连接它们形成 6 位的数，这就是 $f(R_{i-1}, K_i)$，我们将此过程表示为图 4.3。

举个例子来说，假设 $R_{i-1} = 100110$ 和 $K_i = 01100101$，有

$$E(100110) \oplus K_i = 10101010 \oplus 01100101 = 11001111,$$

前 4 位输入到 S_1，后 4 位输入到 S_2，S_1 的第二行、第五列是 000，S_2 的第二行、最后一列

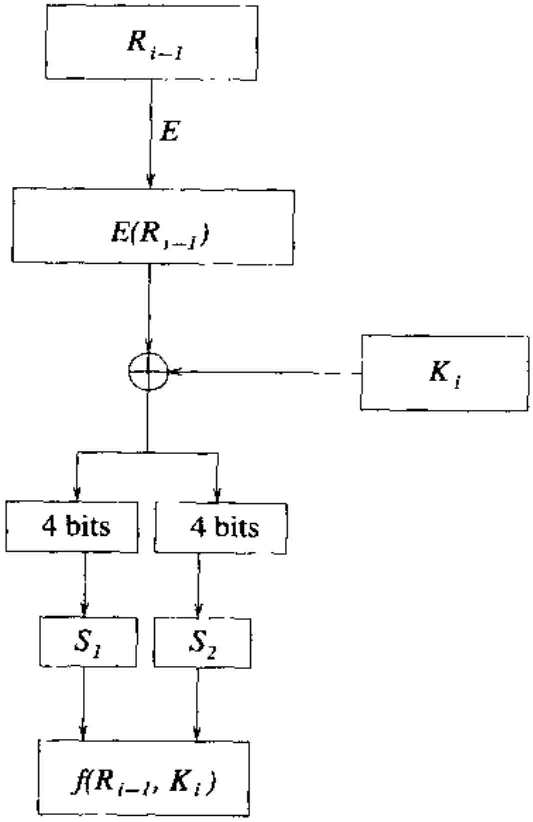


图 4.3　函数 f $(R_{i-1},\ K_i)$

是 100，将这些结果输入函数得 f $(R_{i-1},\ K_i)$ $=000100$。

现在我们可以描述一个循环过程中会发生什么，假设输入是

$$L_{i-1}R_{i-1}=011100100110$$

并且 $K_i=01100101$，在刚才所讨论的例子中 $R_{i-1}=100110$，因此 f $(R_{i-1},\ K_i)$ $=000100$。将它与 $L_{i-1}=011100$ 异或，得出 $R_i=011000$，又由于 $L_i=R_{i-1}$，于是得到

$$L_iR_i=100110011000,$$

将该结果作为下一轮循环的输入。

4.3　微分密码分析法

微分密码分析法是在 1990 年前后由 Biham 和 Shamir 引入并为人们所熟悉的，尽管在此之前 IBM 和 NSA 的 DES 设计者们可能已经知道了它们。该方法的思想就是比较相对应的密文和明文，由此推测出密钥的信息。注意要判断两串字符的不同，可以通过将它们异或得到。由于密钥是通过和 E (R_{i-1}) 异或得来的，查看输入的异或结果，其忽略了在这段信息中密钥的影响，由此移去了密钥中随机的部分内容，这能帮助我们推测出密钥是什么。

4.3.1 具有三轮循环的微分密码分析法

我们最终是要解决在四轮循环下如何攻破上述体制，首先从分析有三轮循环的算法开始，因此，初始设置暂时用 L_1R_1 代替 L_0R_0。

过程描述如下，我们可以利用前面所讲的过程，得到一个有三轮循环的加密设备，而且人们知道加密算法的内部工作过程，如 S-盒，但并不知道密钥，人们希望通过选择明文攻击来找到密钥，使用各种输入 L_1R_1 而得到输出 L_4R_4。

因此有

$$R_2 = L_1 \oplus f(R_1, K_2)$$
$$L_3 = R_2 = L_1 \oplus f(R_1, K_2)$$
$$R_4 = L_3 \oplus f(R_3, K_4) = L_1 \oplus f(R_1, K_2) \oplus f(R_3, K_4)。$$

假设有另一个用 $R_1 = R_1^*$ 表示的消息 $L_1^*R_1^*$，对每一个 i，令 $R_i' = R_i \oplus R_i^*$ 和 $L_i' = L_i \oplus L_i^*$，则 $L_i'R_i'$ 是 L_iR_i 与 $L_i^*R_i^*$ 的"差"（或是和；工作在模 2 方式下）。将前述的计算应用于 $L_1^*R_1^*$，可产生对 R_4^* 的公式。因为已假定 $R_1 = R_1^*$，有 $f(R_1, K_2) = f(R_1^*, K_2)$，因此 $f(R_1, K_2) \oplus f(R_1^*, K_2) = 0$ 和

$$R_4' = R_4 \oplus R_4^* = L_1' \oplus f(R_3, K_4) \oplus f(R_3^*, K_4)。$$

重新整理有

$$R_4' \oplus L_1' = f(R_3, K_4) \oplus f(R_3^*, K_4)。$$

最后，因为 $R_3 = L_4$ 和 $R_3^* = L_4^*$，可得到

$$R_4' \oplus L_1' = f(L_4, K_4) \oplus f(L_4^*, K_4)。$$

注意，如果我们知道了输入的异或即 $L_1'R_1'$，又假设知道了输出 L_4R_4 和 $L_4^*R_4^*$，那么就可以知道最后一个等式中除 K_4 外的任何信息。

现在来分析对 S-盒的输入，以此来计算 $f(L_4, K_4)$ 和 $f(L_4^*, K_4)$。如果从 L_4 开始，首先扩展然后和 K_4 异或，得到 $E(L_4) \oplus K_4$，这是传送到 S_1 和 S_2 的位。同样的，由 L_4^* 可得出 $E(L_4^*) \oplus K_4$。这些异或是

$$E(L_4) \oplus E(L_4^*) = E(L_4 \oplus L_4^*) = E(L_4')。$$

（第一个等式很容易利用扩展函数的逐位描述得来）。因此，可知

1. 两个 S-盒输入的异或（即 $E(L_4')$ 的前 4 位和后 4 位）；
2. 两个输出的异或（即 $R_4' \oplus L_1'$ 的前 3 位和最后 3 位）。

现在来关注 S_1，对 S_2 的分析类似。用给定的所有 4 位输入对进行异或运算（仅指它们当中的 16 位），其速度是相当快的，再寻找哪一些得出了理想的输出异或，这些一旦计算好了，可以将它们存储在一张表中。

例如，假设有输入的异或等于 1011，正在寻找输出的异或等于 100 的值，我们可以运行输入对（1011，0000），（1010，0001），（1001，0010），...，它们每一对异或都等于 1011，并查找输出异或。我们发现（1010，0001）和（0001，1010）产生的输出异或都是 100。例如，1010 意味着查找 S_1 的第二行、第三列，它是 110；进一步，0001 意味着查找第一行、第二列，它是 010，因此输出异或是 $110 \oplus 010 = 100$。

已知 L_4 和 L_4^*，例如，假设 $L_4=101110$ 和 $L_4^*=000010$。因此，$E(L_4)=10111110$ 和 $E(L_4^*)=00000010$，这样输入到 S_1 是 $1011 \oplus K_4^L$ 和 $0000 \oplus K_4^L$，这里 K_4^L 表示 K_4 的左 4 位，如果我们知道对 S_1 的输出异或是 100，那么 $(1011 \oplus K_4^L,\ 0000 \oplus K_4^L)$ 一定是刚才计算过的列表中的一对数，即（1010，0001）和（0001，1010），这意味着 $K_4^L=0001$ 或 1010。

如果重复这个过程多次，应该能去掉 K_4 的两个选择之一，从而确定 K 的 4 位数字。类似地，使用 S_2，可以发现 K 的更多位，因此我们知道了 9 位 K 中的 8 位，最后一位的确定可以通过尝试两种可能性，并看哪一种与我们所攻击的机器具有相同的加密结果，最终即可得到。

将处理过程总结如下（为了叙述简便，我们用同一个 S-盒来描述，虽然在例子中是用各自的 S-盒）：

1. 查看由输入异或 XOR $=E(L_4')$ 和输出异或 XOR $=R_4' \oplus L_1'$ 组成的对列表。
2. 一对数 $(E(L_4) \oplus K_4,\ E(L_4^*) \oplus K_4)$ 在这个列表中。
3. 推测 K_4 的可能值。
4. 重复该过程直到只有一种可能的 K_4。

举例：初始

$$L_1R_1 = 000111011011$$

机器使用密钥 $K=001001101$ 加密了三轮，虽然我们并不知道密钥 K。可以得到（注：因为是从 L_1R_1 开始，我们所用的是移位密钥 $K_1=010011010$）

$$L_4R_4 = 000011100101,$$

如果初始是

$$L_1^*R_1^* = 101110011011$$

（注 $R_1=R_1^*$），那么

$$L_4^*R_4^* = 100100011000,$$

我们有 $E(L_4)=00000011$ 和 $E(L_4^*)=10101000$，输入到 S_1 有异或等于 1010，输入到 S_2 有异或等于 1011，S-盒有输出异或 $R'_4 \oplus L'_1=111101 \oplus 101001=010100$，因此从 S_1 的输出异或是 010，从 S_2 的输出异或是 100。

对一组数（1001，0011），（0011，1001），S_1 产生输出异或等于 010，由于这组数的第一个成员可能是 $E(L_4) \oplus K_4=0000 \oplus K_4$ 的左 4 位，所以 K_4 的前 4 位在 {1001，0011} 之中。对一组数（1100，0111），（0111，1100），S_2 产生输出异或等于 100，由于这组数中的第一个成员可能是 $E(L_4) \oplus K_4=0011 \oplus K_4$ 的右 4 位，故 K_4 的最后 4 位在 {1111，0100} 之中。

现在重复上述过程（使用同样的机器，因此有同样的密钥 K），用

$$L_1R_1 = 010111011011 \text{ 和 } L_1^*R_1^* = 101110011011,$$

类似的分析显示，K_4 的前 4 位在 {0011，1000} 之中，K_4 的后 4 位在 {0100，1011} 之中，结合前面的分析，我们可以得出 K_4 的前 4 位是 0011，而后 4 位是 0100，因此 $K=00*001101$（从 K 的第 4 位开始可以还原 K_4）。

剩下的就是发现密钥 K 的第 3 位了，如果使用 $K=000001101$，它加密 L_1R_1 为 001011101010，它不是 L_4R_4，同时 $K=001001101$ 可以产生正确的密文，因此密钥是 $K=001001101$。

■

4.3.2 具有四轮循环的微分密码分析法

假设我们已经得到了具有四轮循环的设备，而且知道除密钥外的算法的所有内部工作，我们想要确定密钥。我们仍然可以使用前面三轮分析的方法，但将它扩展到四轮需要更多概率论的知识。

S_1 盒有一个弱点，如果使用 16 个异或等于 0011 的输入对，可以发现其中的 12 对输出异或等于 011。当然，平均来说，两组就可产生一给定的输出异或，刚才这种情况是很个别的。可以做一些小的改变，但大的改动将使密钥很容易找到。

S_2 比起 S_1 虽然没有那么多，但也有类似的弱点。在 16 对异或等于 1100 的输入中，有 8 对输出异或等于 010。

假设现在自由地选择初始 R_0 和 R_0^*，以使 $R_0' = R_0 \oplus R_0^* = 001100$。它可扩展为 E (001100) =00111100,因此 S_1 的输入异或是 0011，S_2 的输入异或是 1100。S_1 的输出异或值是 011 的可能性有 12/16，而 S_2 输出异或值是 010 的可能性有 8/16。假设两个 S-盒的输出是各自独立的，可知两者结合的输出异或是 011010 的可能性是 (12/16)(8/16) = 3/8。由于扩展函数分别发送 3 位和 4 位数到盒 S_1 和 S_2，两个盒不可能假定输出是独立的，但 3/8 仍然可认为是对发生事件的相对合理的估计。

现在我们选择 L_0和 L_0^*，使得 $L_0' = L_0 \oplus L_0^* = 011010$。回想在加密算法中，S-盒的输出和 L_0异或得到 R_1，假设 S-盒的输出异或是 011010，那么 $R_1' = 011010 \oplus L_0' = 000000$。由于 $R_1' = R_1 \oplus R_1^*$，有 $R_1 = R_1^*$。

综上所述，如果初始任意选择消息，使异或等于 $L_0'R_0' = 011010001100$，那么大约有 3/8 的可能性是 $L_1'R_1' = 001100000000$。

接下来讨论寻找密钥的策略。尝试几种随意选择的异或等于 011010001100 的输入对，查看输出 L_4R_4 和 $L_4^*R_4^*$。假设 $L_1'R_1' = 001100000000$，用 $L_1' = 001100$ 和已经知道的输出，使用三轮循环微分分析法，去推测一组可能的密钥 K_4。当 $L_1'R_1' = 001100000000$ 时，它可能发生在大约概率 3/8 处，密钥的列表将包括 K_4和一些其他的任意密钥，剩余的 5/8 概率，列表可能包括任意的密钥，因为没有理由认为任意不正确的密钥应该频繁出现，所以正确的密钥 K_4就可能在列表中比其他密钥出现的频率要高。

这儿有一个例子。假设我们正攻击只有四轮的循环设备，尝试 100 对随机输入 L_0R_0 和 $L_0^*R_0^* = L_0R_0 \oplus 011010001100$。可以得到可能的密钥的频率如下表。我们发现很容易分别地查看 K_4的前 4 位和后 4 位。

前 4 位	频率	前 4 位	频率
0000	12	1000	33
0001	7	1001	40
0010	8	1010	35
0011	15	1011	35
0100	4	1100	59
0101	3	1101	32
0110	4	1110	28
0111	6	1111	39

后4位	频率	后4位	频率
0000	14	1000	8
0001	6	1001	16
0010	42	1010	8
0011	10	1011	18
0100	27	1100	8
0101	10	1101	23
0110	8	1110	6
0111	11	1111	17

由表得知，很可能 $K_4 = 11000010$。因此，密钥 K 是 10 * 110000。

要确定剩余的 1 位，采用前面的处理方式，我们可以将信息 000000000000，用 $K = 101110000$ 加密成 100011001011，用 $K = 100110000$ 加密成 001011011010。如果所攻击的机器加密 000000000000 成为了 100011001011，得出第二个密钥是不正确的，所以正确的密钥可能是 $K = 101110000$。

通过扩充这些方法，可以将攻击过程扩展到更多轮的循环加密中。你可能已经注意到了，可以简单地通过运行所有可能的密钥，很快得出密钥，在简单的模型中这种结果是正确的，但在更精细的体制如 DES 中，微分分析法的技术就比穷尽搜索所有的密钥效率要高得多，尤其对循环轮数比较多的体制。特别强调，DES 使用 16 轮循环，很明显用微分分析法比穷尽搜索效率更高，所以往往 16 轮以上的加密算法用微分分析法来破译。

破解 DES 的另外一种方法叫做**线性密码分析法**（**Liner cryptanalysis**），是由 Mitsuru Matsui [Matsui] 提出的。它的主要要素是一个 DES 的输入位的线性函数的近似值，在理论上比穷尽搜索密钥法要快许多，它大约需要 2^{43} 个明文 - 密文对就能找到密钥。但看起来 DES 的设计者并未预测到线性密码分析法。算法的详细内容见 [Matsui]。

4.4 DES

一组密文由 64 位组成，密钥是 56 位，但它是用 64 位的字串表示的。第 8 位，第 16 位，第 24 位，... 是奇偶位，这样的安排是为了使每 8 位一组中都有一个奇数 1，这是出于错误检测的目的，加密的输出是 64 位的密文。

DES 算法如图 4.4 所示，初始用 64 位的明文 m 表示，主要有三步。

1. m 的各位置通过初始变换改变其排列顺序得到 $m_0 = IP(m)$，记为 $m_0 = L_0R_0$，这里 L_0 是 m_0 的前 32 位，R_0 是 m_0 的后 32 位。

2. 对 $1 \leq i \leq 16$，执行如下操作：

$$L_i = R_{i-1}$$
$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i),$$

K_i 是来自于密钥 K 的一个 48 位的字串，f 是函数，后面会讲到。

3. 左右交换得到 $R_{16}L_{16}$，然后执行初始变换的逆变换，得到密文 $c = IP^{-1}(R_{16}L_{16})$。

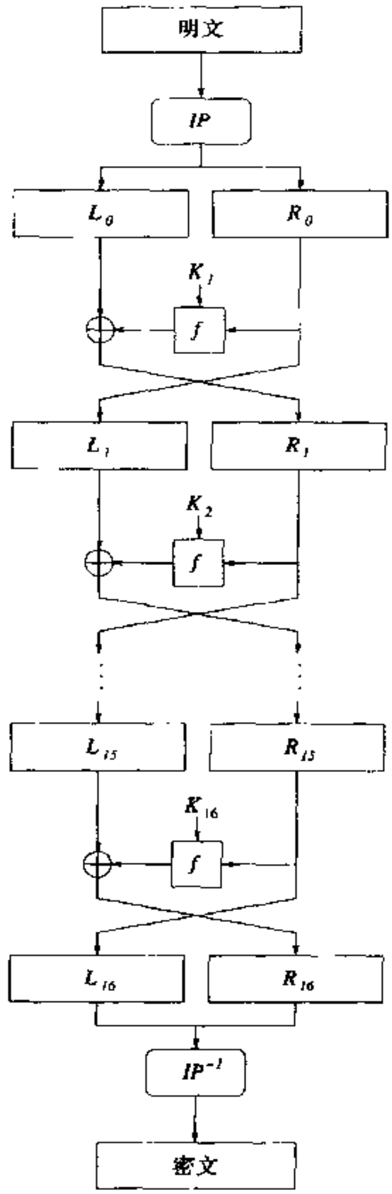


图 4.4 DES 算法

解密过程与加密过程完全相同，只是按密钥 K_1，...，K_{16}相反的顺序而已，具体的原因与 4.2 节简单体制描述中的过程是相同的。注意 DES 算法第三步中的左右交换表明，我们并不是必须进行左右交换，尽管在 4.2 节中对解密来说这种交换是需要的。

现在来详细描述每一步。

可由初试置换表描述的初试置换计算，似乎没有太多的密码重要性，但其设计的目的可能是便于在 20 世纪 70 年代的芯片中更好地运行。这表明 m 的第 58 位变成 m_0的第 1 位，m 的第 50 位变成 m_0的第 2 位，等等。

初始置换表															
58	50	42	34	26	18	10	2	60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6	64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1	59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5	63	55	47	39	31	23	15	7

函数 $f(R, K_i)$ 如图 4.5 所示，用如下几步进行描述：

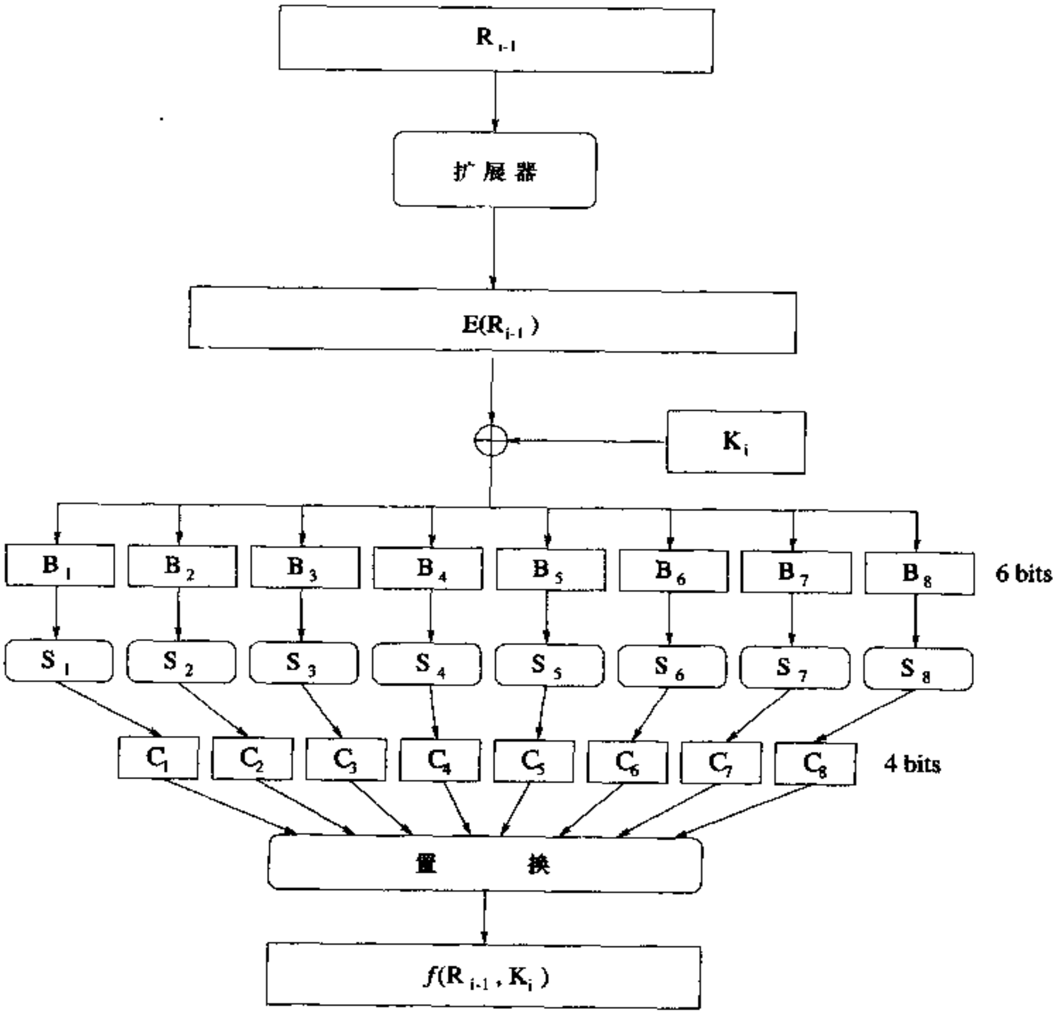


图 4.5 DES 的函数 $f(R_{i-1}, K_i)$

1. 首先，将 R 按下表扩展成 $E(R)$。

扩展置换表											
32	1	2	3	4	5	4	5	6	7	8	9
8	9	10	11	12	13	12	13	14	15	16	17
16	17	18	19	20	21	20	21	22	23	24	25
24	25	26	27	28	29	28	29	30	31	32	1

这意味着 $E(R)$ 的第 1 位是 R 的第 32 位，等等。注意 $E(R)$ 有 48 位。

2. 计算 $E(R) \oplus K_i$，它有 48 位，记作 $B_1B_2 \ldots B_8$，这里每一个 B_j 有 6 位。

3. 有 8 个 S-盒，即 $S_1, \ldots, S_8$，在下一页中给出，B_j 是 S_j 的输入，记作 $B_j = b_1b_2 \ldots b_6$。b_1b_6 确定 S-盒的行，$b_2b_3b_4b_5$ 确定 S-盒的列。例如，如果 $B_3 = 001001$，查看行 01，它就表示第二行（00 表示第一行），查看列 0100，表示第五列（0100 代表二进制 4；第一列用数字 0 表示，故第五列用 4 表示）。S_3 盒中在该位置的值是 3，表示的是二进制的 3，因此本例中 S_3 的输出是 0011。按此方式，可以得到 8 个 4 位的输出 $C_1, C_2, \ldots C_8$。

4. 字串 $C_1C_2 \ldots C_8$ 根据下表进行置换。

16	7	20	21	29	12	28	17	1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9	19	13	30	6	22	11	4	25

结果，32 位的字符串就是 $f(R, K_j)$。

最后，我们讲述怎样得到 $K_1, \ldots K_{16}$。回想一下初始是用 64 位的 K。

1. 去掉奇偶位，其余位按下列表的顺序置换。

密钥置换表													
57	49	41	33	25	17	9	1	58	50	42	34	26	18
10	2	59	51	43	35	27	19	11	3	60	52	44	36
63	55	47	39	31	23	15	7	62	54	46	38	30	22
14	6	61	53	45	37	29	21	13	5	28	20	12	4

将结果记为 C_0D_0，这里 C_0 和 D_0 各有 28 位。

2. 对 $1 \leqslant i \leqslant 16$，令 $C_i = LS_i(C_{i-1})$ 和 $D_i = LS_i(D_{i-1})$，这里 LS_i 表示按下表左移输入一到两个位置。

每轮循环密钥位移动的位数																
循环轮数	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
移位	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

3. 从 56 位字串 C_iD_i 中按下表选取 48 位，输出是 K_i。

14	17	11	24	1	5	3	28	15	6	21	10
23	19	12	4	26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40	51	45	33	48
44	49	39	56	34	53	46	42	50	36	29	32

可以知道，密钥的每一位是用 16 轮中的大约 14 位生成的。

有一些其他的观点，在好的密码体制中，密文中的每一位都应该依赖于明文中的所有位。扩展函数 $E(R)$ 被设计成仅和某些轮的数据有关，最初的这种转换人们并不完全清楚，这也不具有密码分析的意义。S-盒是算法的核心并提供了安全性，它们的设计一直有些神秘，直到 20 世纪 90 年代早期，IBM 颁布下列标准（详细内容见 [Coppersmith1]）。

S-盒

S-box 1

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S-box 2

15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S-box 3

10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S-box 4

7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S-box 5

2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S-box 6

12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S-box 7

4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S-box 8

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

1. 每一个S-盒都有6位输入和4位输出。在1974年，这是设计在一块芯片上的最大一部分内容。

2. S-盒的输出不要接近输入的线性函数（线性可使系统很容易分析）。

3. S-盒的每一行包括0～15的所有数字。

4. 如果针对一个S-盒的两次输入有1位不同，那么这两次输出至少要有2位不同。

5. 如果针对一个S-盒的两次输入前两位不同，但后两位相同，那么输出一定要不相同。

6. 给定了32对输入的异或值。对每一对，计算其输出的异或，这些输出异或至多有8个相同，很显然，这是为了避免通过微分密码分析来攻击。

7. 与标准（6）类似，但针对的是3个S-盒。

在20世纪70年代早期，用一台计算机计算相应的S-盒需要几个月的时间，现在，同样的计算只需要很短的时间。

DES不是一个群

有效地增加DES密钥大小的可能方法就是双重加密。选择密钥K_1和K_2并通过$E_{K2}(E_{K1}(P))$加密明文P，这能增加安全性吗？

密码体制中基于中间相遇攻击（Meet-in-the-middle）在8.4节中进行讨论，这部分内容指出，如果攻击者有足够的内存，双重加密几乎不可能提供什么额外的保护。而且，如果一个双重加密体制等同于单一加密体制的话，那么通过双重加密得不到额外的安全保护。

此外，如果双重加密等同于单一加密，那么这个（单一）加密体制比我们最初的想象（见练习8.10）有更少的安全性。例如，如果这些对DES成立的话，就可以用更容易实现的检索长度大约是2^{28}来代替穷尽检索所有2^{56}个密钥的方法。

对仿射密码（见2.2节）和RSA（第6章），用两个密钥K_1和K_2的双重加密等同于用第三个密钥K_3的加密，那么对DES也是这样吗？即，能有一个密钥K_3满足$E_{k3}=E_{k2}E_{k1}$吗？这个问题经常用另一个等价的形式“DES是一个群吗？”来描述（不熟悉群论的读者可以说“DES在代数运算下是封闭的吗？”）。

幸运的是，可以证明DES不是一个群。这里略去证明。更详细的内容见［Campbell-Wiener］。设E_0表示全部由0组成的密钥，E_1表示全部由1组成的密钥，这些密钥在密码学思想中是很脆弱的（见练习5），而且，D. Coppersmith发现对某一明文重复应用$E_1 o E_0$，经过大约2^{32}次反复后就产生最初的明文。加密序列如下（对明文P）

$$E_1E_0(P), E_1E_0(E_1E_0(P)), E_1E_0(E_1E_0(E_1E_0(P))), \ldots, (E_1E_0)^n(P)=P,$$

n是满足$(E_1E_0)^n(P)=P$的最小正整数，称为循环长度n。

引理：如果m是满足对所有P有$(E_1E_0)^m(P)=P$的最小正整数，n是某一循环长度（满足对某一P_0有$(E_1E_0)^n(P_0)=P_0$），那么m可以被n整除。

证明：设m除以n余数为r，这表明对某些整数q，$m=nq+r$，$0\leqslant r<n$，因为$(E_1E_0)^n(P_0)=P_0$，用$(E_1E_0)^n$加密q次，剩下的P_0仍未改变。因此，

$$P_0=(E_1E_0)^m(P_0)=(E_1E_0)^r(E_1E_0)^{nq}(P_0)=(E_1E_0)^r(P_0)$$

由于n是满足$(E_1E_0)^n(P_0)=P_0$的最小正整数，$0\leqslant r<n$，一定有$r=0$，这意味着$m=nq$，这样m可以被n整除。证毕。 □

现在假设DES在代数运算下是封闭的，那么对某些密钥K有$E_1E_0=E_k$，进而，E_k^2，

E_K^3，... 也可表示 DES 密钥。由于仅有 2^{56} 种可能的密钥，对某些整数 i，j，$0 \leqslant i < j \leqslant 2^{56}$，一定有 $E_K^j = E_K^i$（否则就有 $2^{56}+1$ 个不同的加密密钥）。第 i 次解密：$E_K^{j-i} = D_K^i E_K^j = D_K^i E_K^i$，它们具有相同的结果。因为 $0 < j - i \leqslant 2^{56}$，与 E_K^m 有相同结果的最小正整数 m 也满足 $m \leqslant 2^{56}$。

Coppersmith 利用 33 个明文 P_0 发现了循环长度，由引理，m 是这些循环长度的倍数，因此，m 大于或等于这些循环长度的最小公倍数，可以证明大约是 10^{277}。但如果 DES 在代数运算下是封闭的，可证明 $m \leqslant 2^{56}$，所以 DES 在代数运算下是不封闭的。

4.5 操作模式

DES 是分组加密算法的一个特例，在 DES 中一组明文是 64 位，将它加密成了一组密文。这些算法可以运行在许多不同的模式下，下面介绍三种最流行的操作模式。

4.5.1 电子密码本（ECB）

使用分组密码很自然的方式就是将一长串明文分解成为适当的分组，对每一分组用加密函数 E_k（ ）分别加密。这就是众所周知的电子密码本（ECB）操作方式。明文 P 被分解为小一点的分组 $P = [P_1, P_2, ..., P_L]$，其对应的密文是

$$C = [C_1, C_2, ..., C_L]$$

这里 $C_j = E_K(P_j)$ 是明文 P_j 使用密钥 K 加密的结果。

ECB 操作模式固有的缺点在明文很长的情况下变得更明显了，当攻击者伊芙长时间地一直观察艾丽斯和鲍勃之间的通信时，如果伊芙想方设法获得了一些她观察到的明文及相应的密文，她就可以开始建立电子密码本，进而译出艾丽斯和鲍勃后续的通信。伊芙不必要去计算密钥 K；她只需查看其电子密码本上的密文信息，并用对应的明文（如果有的话）去破解信息。

由于现实世界中的信息是由许多重复的段落组成的，这个问题就变得相当严重了。E-mail 是首要的例子，艾丽斯和鲍勃之间的 e-mail 可能用如下字头开始：

Date: Tue,29 Feb 2000 13:44:38 -0500 (EST)

密文用 8 个字符的加密版本开始“Date: Tu”。如果伊芙发现这段密文经常出现在星期二，她不必知道任何明文信息，就可能猜出这段信息是星期二发送的 e-mail。靠着她的耐心和创造性，伊芙可能将这些足够的信息头组合在一起，并最终发现信息的内容。使用更大的耐心和计算机内存，她能够组合信息中更重要的内容。

ECB 模式的另外一个问题当伊芙尝试修改发送给鲍勃的加密信息时会发生。她能够抽取信息的重要部分，并使用她的电子密码本去产生一个错误的密文信息，她可以将其插入数据流通信中。

4.5.2 密码分组链（CBC）

减少 ECB 模式存在的问题的一种方法是使用链接。链接是一种反馈机制，一块分组的

加密依赖于其前面分组的加密。具体地说，加密过程是：

$$C_j = E_K(P_j \oplus C_{j-1})$$

而解密过程是：

$$P_j = D_k(C_j) \oplus C_{j-1}$$

C_0是某个选定的初始值，D_K（ ）是解密函数。

由此可见，在 CBC 模式中，明文和前一分组的密文异或后，再将其结果进行加密，图 4.6 描述了 CBC。

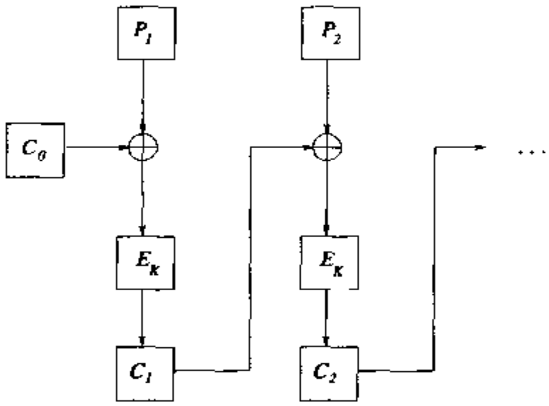


图 4.6　密码分组链接模式

4.5.3　密码反馈（CFB）

CBC 模式的问题之一是，即使明文错一位或在计算/存储以前的密文分组中有一点错误，都可能导致当前密文组的计算错误，这必将影响所有后续的密文组。CBC 和 ECB 方法都有的另一个缺点是，在完整的 8 字节的数据分组未到来之前，加密（和解密）是不能开始的。

密码反馈模式是一种流操作模式，一组 8 位信息（如一个字符）并不需要等待全部的数据分组到达后才能加密，在交互式的计算机通信中这是非常有用的，举一个例子。

将明文分成 8 位一组：$P = [P_1, P_2, \ldots]$，这里每一个 P_j都是 8 位，而不像在 ECB 和 CBC 中使用的是 64 位。加密过程如下，选择初始的 64 位 X_1，对 $j=1, 2, 3, \ldots$，执行下列过程：

$$C_j = P_j \oplus L_8(E_k(X_j))$$
$$X_{j+1} = R_{56}(X_j) \parallel C_j。$$

L_8（X）表示 X 最左边的 8 位，R_{56}（X）表示 X 最右边的 56 位，$X \parallel Y$ 表示 Y 紧接 X 后所组成的字符串。

解密用如下步骤：

$$P_j = C_j \oplus L_8(E_k(X_j))$$
$$X_{j+1} = R_{56}(X_j) \parallel C_j。$$

让我们来执行一轮 CFB 算法。首先，有一个初始的 64 位注册器用 X_1表示，这 64 位用 E_k加密，并取 E_K（X_1）最左边的 8 位异或 8 位的 P_1形成 C_1，然后将 C_1传送至接收器。在

P_2工作之前，64 位的注册器 X_1 修改为最右边的 56 位，将 8 位 C_1 放在右边形成 $X_2 = R_{56}(X_1) \parallel C_1$。然后按同样的过程加密 P_2，但用 X_2 代替 X_1。P_2 被加密成 C_2 后，64 位注册器被修改形成

$$X_3 = L_{56}(X_2) \parallel C_2 = L_{48}(X_1) \parallel C_1 \parallel C_2。$$

一直到第 8 轮结束，最初来自 64 位注册器的 X_1 也已消失，$X_9 = C_1 \parallel C_2 \parallel \cdots \parallel C_8$。$C_j$ 继续通过注册器，一直到如 $X_{20} = C_{12} \parallel C_{13} \parallel \cdots \parallel C_{19}$。

注意 CFB 加密明文的方式类似于一次一密方法和 LFSR。密钥 K 和数字 X_j 用来产生二进制串，再将其与明文异或生成密文。这与 ECB 和 CBC 是非常不同的加密形式，它们的输出是 DES 密文。

在实际应用中，CFB 是很有用的，因为它可以从密文传输的错误中恢复过来。假设发送者发送了密文组 C_1，C_2，...，C_k，...，并且 C_1 在传输中被破坏，这样接收方收到了 $\tilde{C}_1$，C_2，...。由于 $\tilde{C}_1$ 出现了个别位的错误，解密 $\tilde{C}_1$ 就可能出现不正确的 P_1 明文。现在，解密这个密文组后，接收者形成了不正确的 X_2，用 $\tilde{X}_2$ 表示，如果 X_1 是（*，*，*，*，*，*，*，*），那么 $\tilde{X}_2$ =（*，*，*，*，*，*，*，$\tilde{C}_1$）。即使接收者收到了正确的 C_2 并解密它，也将产生完全混淆的 P_2 版本。当形成 X_3 时，解密者实际上形成了 $\tilde{X}3$ =（*，*，*，*，*，*，$\tilde{C}_1$，C_2），解密者重复此过程，最终得到了错误的 P_1，P_2，...，P_9 版本。当解密者计算 X_9 时，错误的分组已经移到了 $\tilde{X}_9$ 分组的最左边，$\tilde{X}_9$ =（$\tilde{C}_1$，C_2，⋯，C_8）。在下一步，这个错误将从 X_{10} 注册器中消失，并且 X_{10} 及后续注册器不会出现混淆错误。

4.6 破解 DES

DES 在 20 世纪最后 20 年一直作为密码学标准的加密体制，在历史上它多少带有一些神秘色彩；更多的问题集中在人们认为 NSA 设置了或强或弱的安全问题。人们花费了大量的时间和精力寻找 NSA 可能安装在算法中的陷门和漏洞，虽然 DES 的真相不为人所知（除非等到 100 年后大量的政府机密文件公诸于世），但它已展示出了超强的抗击蛮力攻击能力，同时也具有抵御精密复杂的攻击的能力，如微分密码分析。

近来，DES 已经开始暴露出它的“老态”。本节我们将讨论在攻破 DES 方面的各种努力，并且现在很可能要选择将 DES 淘汰出历史舞台。

1975 年以前，一直有人质疑 DES 的抗攻击性，许多学术机构团体抱怨 DES 密钥的长度不够，声称 56 位密钥是不安全的。事实上，NBS 公开 DES 几个月后，Whitfield 和 Martin Hellman 发表了一篇题为“NBS 数据加密标准详尽密码分析”的论文，论文中他们估计用 2 千万美元（1977 年的美元标准）建立的设备花大约一天的时间就会破解 DES。这种设备的目的就是专门用来破译 DES 的，关于这一点我们后面还要讲到。

到 1987 年 DES 需要承受第二个五年审查。在这段时期，NBS 就是否接受它作为下一期的标准还是修改标准或溶合其他标准给出了一个意见，针对 DES 的讨论看出，NSA 反对给 DES 换发新证，在当时，NBS 提出 DES 已开始显现出弱点的信号，就当时的计算水平而言，建议全部废除 DES，采用一套 NSA 新设计的算法代替之，算法内部的工作过程只有 NSA 知

道，这可以更好地从工程技术的方面保护它。这个建议最终未被采纳，部分原因在于几个主要的美国公司在置换算法提出期间利益得不到保护。最后，重新提出把 DES 作为标准，但在讨论的过程中，人们已经认识到 DES 存在弱点。

5 年以后，NBS 重新命名为 NIST，下一个五年审查又到来了。尽管在 1987 年其弱点就已暴露出来了，而且 5 年内技术发生了进步，但 NIST 在 1992 年还是重新确认了 DES 算法。

1993 年，工作在 Bell-Northern 研究所的研究员 Michael. Wiener 提出并设计了一种设备，它比以往任何一种设备都能更高效地攻破 DES。他的这种思想就是已经广泛使用在电话业中的开关技术。1996 年，出现了攻击对称密码如 DES 的 3 种方法的详细说明。第一种方法是在一个有大量情报的机器上做分布式计算，它的优点是相对便宜，费用也易于分配。另一种方法是设计自定义的体系结构（如 Michael. Wiener 的思想）来破译 DES，一般来说，它的效率较高，但费用也很昂贵，可以认为是最尖端的方法。另一种折衷的方法是采用可编程逻辑阵列，它受到了一定的关注。

在所有 3 种方法中，最流行的破译 DES 的方法是对 DES 密钥空间进行穷尽搜索。对 DES 来说这似乎相当困难，如前面所讲述的，更复杂的密码分析技巧都已失败，这表明穷尽搜索需要进行重大改变。

用分布式计算方法来破译 DES 越来越受人喜爱，特别是随着 Internet 的普及，1997 年 RSA 数据安全公司组织了一次具有挑战性的活动，寻找密钥并破解 DES 加密信息，谁破解了这个信息就将赢得 1 万美元的奖金，仅仅在 1997 年 DES 挑战赛宣布 5 个月后，Rocke Verser 就提交了正确的 DES 密钥。这一成果标志着分布式计算已经可以成功地攻破 DES。Rocke Verser 将已编制的程序发布在 Internet 成千上万台机器上，共同运行来破译 DES 密码，许多人或团体自愿贡献他们的机器，运行 Verser 的程序，Verser 承诺将拿出其奖金的 60% 到 40% 给计算机的所有者，事实上真的找出了密钥，密钥的最终发现者是 Michael Sanders，大约在运行了 DES 密钥空间的 25% 的时候发现了密钥。DES 挑战赛短语解密为“强壮的密码体制使世界更安全了”。

在接下来的一年里，RSA 数据安全委员会组织了第二届 DES 挑战赛，这次正确的 DES 密钥是由分布式计算技术发现的，而且消息被解密为“很多双手使工作变得轻松”。在经历了 39 天，大约搜索了 85% 可能的密钥空间后，成功地找到了密钥。第二届比赛的获胜者搜索了比第一届更多的密钥空间，且执行的速度更快，这表明，一年来技术上的进步对密码分析产生了戏剧性的影响。

1998 年夏季，电子尖端基金会（EFF）设立一个工程叫 DES 解密高手，它的目的就是针对一种特殊的体系结构，暴露 DES 算法的弱点。DES 解密高手工程的设立基于简单的原理：一般的计算不能很好地承担破译 DES 的任务，这也是很正常的，因为它们固有的属性就是适用于多用途的机器，用来完成普通的任务，如运行操作系统，甚至是用来玩计算机游戏，或两者兼而有之。EFF 小组计划设计一种特殊的硬件，它们具有并行穷尽搜索这种性质的优点。小组的经费预算有 20 万美元。

现在我们简单地描述一下 EFF 小组研究生产的这种体系结构。关于 EFF 解密高手及其他处理方案的更多信息，见［Gilmore］。

EFF DES 解密高手由 3 个重要部分组成：个人计算机、软件和大规模集成芯片。计算机与芯片阵列连接在一起，软件只检查每一个芯片的工作，大部分情况下，软件不和硬件直接

交互；它仅给芯片初始处理必需的信息，并等待，直到芯片返回候选密钥。在这种情况下，硬件有效地删除了大量无效的密钥，它仅返回可能的潜在密钥，然后软件再处理它以得到可能的潜在密钥，检查这些可能的密钥是否是最终正确的密钥。

DES 解密高手选取了 128 位（16 字节）的明文，并将它分成两个 64 位的分组。EFF DES 解密高手中的每个芯片由 24 个搜索单元组成，每一个搜索单元是芯片的子集，芯片的任务就是利用一个密钥和两个 64 位密文组，尽量解密利用这个密钥加密的第一个 64 位分组。如果这个“破译的”密文看起来有意义，那么搜索单元解密第二个分组，并检查这个“破译的”密文是否也是有意义的，如果两个破译的密文都是有意义的，搜索单元就会告诉软件它检测的密钥有可能是正确的。如果当第一个 64 位密文分组被解密后，解密的密文似乎并不足够有意义，那么搜索单元会将密钥加 1，形成一个新的密钥，并按同样的方式处理，直到搜索完分配给它的全部密钥空间。

EFF 小组是怎样定义“有意义”（interesting）的解密文本的呢？首先他们假设明文满足某些基本的假设，如它们是用字母、数字和标点符号组成的。因为需要解密的信息是文本，他们知道每个字节对应一个 8 位的字符。当然一个 8 位的字符有 256 种可能的取值，仅有 69 个字符是有意义的（大写和小写字母、数字、空格和一些标点符号）。对于一个被认为有意义的字节，它应该包含这 69 个字符之一，因此有意义的概率是 69/256。近似于 1/4，并假定解密字节之间是相互独立的，于是一个 8 字节解密分组是有意义的概率是 $1/4^8=1/65536$。这样仅需要检查密钥空间的 1/65536，才认为是有可能的密钥。

这减少得还不够，软件仍要花费大量的时间搜索错误的候选密钥，为了进一步缩小可能的候选密钥的范围，需要使用第二个 8 字节的明文分组，这个分组也被解密，看结果是否是有意义的。假设两分组是独立的，我们可以得出：仅有 $1/4^{16}=1/65536^2$ 密钥被认为是可能的密钥。这大大减少了软件需要检测的密钥空间的数量。

每个芯片是由 24 个搜索单元组成的，每个搜索单元又给出了它相应要搜索的密钥空间。一个 40MHz 的芯片要花费大约 38 年才能检索完全部的密钥空间。为了进一步减少需要处理的密钥的数量，EFF 小组在一块线路板上使用 64 个芯片，每个底盘上有 12 块线路板，最终有两个底盘连接到用软件检查通信的个人电脑上。

最终结果是 DES 解密高手大约由 1500 个芯片组成，破解 DES 平均用了大约 4.5 天。DES 解密高手对破解 DES 来说决不是最好的模式。特别是每一个芯片运行速度是 40MHz，用现在的标准看太慢了。新的模式当然可以在将来生产出来，这样芯片就可以运行在非常快的时钟周期上了。

随着技术不断发展，人们强烈地感到需要替换 DES 算法。有两种主要的方法可用来进一步增加 DES 的安全性。第一种方法是多次使用 DES，由此产生了广为喜爱的方法，称之为三重 DES。第二种方法是寻找一种新的体制，它比 56 位密钥大许多。

现在来介绍一下三重 DES 的设计思想。其基本思想就是使用同样的算法，用不同的密钥加密三次相同的密文。**双重加密（Double DES）**是第一次用一个密钥加密明文，然后再使用不同的密钥加密一次。由于 DES 并未形成一个群（见 4.4 节），人们可能会猜测双重 DES 应该用双倍的密钥空间，这样密钥空间就由 2^{112} 个密钥组成。当然这并不正确。Merkle 和 Hellman 已证明双重加密设计事实上具有 57 位密钥等级的安全性，使用从**中间相遇攻击**（见 8.4 节）密钥空间从 2^{112} 减少到 2^{57}。

因为双重 DES 有它的弱点，经常使用的是三重 DES，很明显它具有近似等于 112 位密钥的加密级别的安全性。至少有两种方式来实现三重 DES，一种是选择 3 个密钥 K_1，K_2，K_3并执行 E_{K1}（E_{K2}（E_{K3}（m）））。另一种是选择两个密钥 K_1 和 K_2 并执行 E_{K1}（D_{K2}（E_{K1}（m）））。当 $K_1 = K_2$时，这减少为简单 DES。考虑到兼容性，在中间使用 D_{K2}代替 E_{K2}；使用 D 代替 E 没有什么特别的增强加密。两种版本的三重 DES 都能抵抗来自中间相遇的攻击（比较练习 6）。但也有其他针对双重密钥版本的攻击（［Merkle-Hellman］和［van Oorschot Wiener］），由于它们要求太大的内存空间以致于不可行，这表明它们有一些可能的弱点。

另外一种加强 DES 的方法由 Rivest 提出，选择 3 个密钥 K_1，K_2，K_3并执行 $K_1 \oplus E_{K2}$（$K_3 \oplus m$）。换句话说，明文通过与 K_1异或，然后用 K_2应用 DES，最后将结果与 K_3异或。这种方法称为 DESX，已经证明其相当安全。见［Kilian-Rogaway］。

另外，在加密算法家族中还发展了一种新的算法。在 1998 年 NIST 提出了 15 个候选加密算法，用它取代 DES 作为新的加密标准，这就是众所周知的高级加密标准（AES）。在 2000 年，其中之一 Rijndael 被选作 AES，下一章将对它进行讲述。

4.7　口令的安全

当你登录计算机并输入口令时，计算机检查你输入的口令是否属于这个用户并授权使用，但是将口令存储在计算机的文件中是相当危险的，得到文件的人可能打开任何人的账户。使口令文件仅对计算机的管理员可存取是一种解决办法，但如果管理员在换工作之前对文件做了备份，会发生什么事情？因此根本的解决办法是在存储口令之前对它进行加密。

令 f（x）是**单向函数（one-way function）**，表明很容易计算 f（x），但很难解 $y = f$（x）中的 x。口令 x 可以作为 f（x）存储，连同用户名一起。当用户登录的时候，输入口令 x，计算机计算 f（x）并检查与这个用户相匹配的 f（x）的值，得到了口令文件的入侵者也只有每个用户 f（x）的值，为了登录到其帐号下，入侵者需要知道 x，这是很难计算的。

在许多系统中，加密口令是存储在公共文件下的，因此，访问系统的任何人都可以得到这个文件。假设函数 f（x）已知，那么字典中的所有单词和这些单词的变形（比如反过来写它们）都可以输入到 f（x）中。用口令文件与这些结果作比较，就经常可以得到几个用户的口令。

通过使口令文件不公开可以部分阻止这种**字典攻击（dictionary attack）**，但仍然会存在不遵守规章的（或过激的）计算机管理员问题，因此，还需要其他的使信息更安全的方式。

还有另外一个有趣的问题，若 f（x）很快地计算出来了，这似乎是可以做到的，然而，稍微减慢 f（x）的计算速度可以降低字典攻击的速度，但如果将 f（x）计算减慢太多就会引起问题。如果 f（x）设计运行在速度非常快的计算机上是 1/10 秒，那么如果在一个很慢的计算机上登录会耗费令人不能接受的大量时间，这样解决显然不令人十分满意。

阻止字典攻击的一种方式是用所说的 salt，每个口令用附加的 12 位数自由填充。这些 12 位数用来修改函数 f（x），结果存储在口令文件中，同时还存储了用户名和 12 位 salt 值。当用户输入口令 x 时，计算机在此文件中寻找该用户的 salt 值，然后利用之计算 f（x），将结果与存储在文件中的值进行比较。

当使用 salt 值并且字典中的单词输入到 $f(x)$ 中时，他们需要用 $2^{12}=4096$ 种可能的 salt 值填充，这相当大地降低了运算速度，而且假设攻击者存储了字典中所有单词的 $f(x)$ 值，这可以预料能攻击几个不同的口令文件。用 salt 值，对存储空间的要求显著增加，因为每个单词需要存储 4096 次。

salt 值的主要目的是阻止针对发现任何人口令的攻击。特别地，它使那些很弱的可选择口令变得安全，因为很多人使用弱的口令，salt 值在这种情况下使用最理想。salt 并没有降低针对个人口令的攻击速度（除非禁止使用未经授权的 DES 芯片）。如果伊芙想要发现鲍勃的口令并访问口令文件，她发现了鲍勃使用的 salt 值，并尝试了一次字典攻击，比如，使用惟一的 salt 值，如果鲍勃的口令不在这个字典中，那么就会失败，伊芙不得不对所有可能的口令进行穷尽搜索。

在许多 Unix 口令规则中，单向函数是建立在 DES 基础上的，口令的前 8 个字符被转换成 7 位 ASCII 码（见 2.8 节），这 56 位成为了 DES 密钥。如果口令少于 8 个符号，用 0 填充，得到 56 位。然后所有是 0 的“明文”利用该密钥采用 DES 的 25 轮加密。输出存储在口令文件中。函数

$$\text{password} \rightarrow \text{output}$$

被确认是单向的，即已知“密文”（是输出）和“明文”（全是 0）。寻找的密钥就是口令，相当于对 DES 做已知明文的攻击，这一般来说是很困难的。

为了增加安全性，salt 按下列方式增加，产生一个随机的 12 位数作为 salt 值，回想前述的 DES，扩展函数 E 将 32 位的输入 R（对一轮循环输入的右边）扩展成 48 位 $E(R)$。如果 salt 值第 1 位是 1，将第 1 位与 $E(R)$ 的第 25 位交换，如果 salt 值的第 2 位是 1，交换第 2 位和 $E(R)$ 的第 26 位，继续至 salt 值的第 12 位，如果是 1，第 12 位与 $E(R)$ 的第 36 位交换，当 salt 值的某一位是 0，不需要交换。如果 salt 值全部为 0，则不会发生交换，就是工作在通常所说的 DES 下。这种情况下，salt 意味着 4096 种 DES 的可能。

使用 salt 修改 DES 的一个优点是，在执行字典攻击时，人们不能用高速的 DES 芯片计算单向函数。相反，设计的芯片要尝试所有 4096 种由 salt 值引起的 DES 的变化。否则攻击只能用软件执行，速度要慢许多。

许多人认为 salt 是一种临时的方法，随着存储空间的增加及计算机速度的改进，4096 种情况已不是什么要紧的因素。由于这个原因，在进一步的实施中几个新的口令方案正被研究。

4.8 习　题

1. 考虑下列类 DES 加密方法。原始信息有 $2n$ 位，划分成长度为 n 的两组（左右各一半）：M_0M_1。密钥 K 由 k 位组成，k 为任一整数。函数 $f(K, M)$ 输入分别是 k 位和 n 位，输出为 n 位。每一轮加密用一对 M_jM_{j+1} 作为初始，输出对是 $M_{j+1}M_{j+2}$，这里

$$M_{j+2}=M_j \oplus f(K, M_{j+1})。$$

（$\oplus$表示异或，它是指每位模 2 加法）。这要执行 m 轮，因此密文是 M_mM_{m+1}。

(a) 如果有一台机器按上述方式进行 m 轮加密，怎样使用同样的机器解密密文 M_mM_{m+1}

(使用同样的密钥)? 请解释原因。

(b) 假设 K 有 n 位并且 $f(K, M)=K \oplus M$，假设加密过程由 $m=2$ 轮组成，如果仅知道密文，你能推测出明文和密钥吗？如果你知道密文及相应的明文，你能推测出密钥吗？请解释原因。

(c) 假设 K 有 n 位并且 $f(K, M)=K \oplus M$，假设加密过程由 $m=3$ 轮组成，为什么这体制是不安全的？

2. 如 4.7 节所述，计算机中存储口令的普通方式是使用 DES，用口令作为密钥加密一个固定的明文（通常 000...0）。将密文存储在文件中，当你登录时，重复此过程并比较密文。为什么这种方法比使用口令作为明文和使用固定密钥（如 000...0）的类似方法更安全？

3. 证明用 CBC 和 CFB 模式的解密过程实际上得到的是有效的解密。

4. 对一位串 S，令 $\overline{S}$ 表示互补串，它是通过将所有 1 转变成 0、所有 0 转变成 1 得到的（等价于 $\overline{S}=S \oplus 11111...$）。证明，如果在 DES 算法中，用密钥 K 加密 P 成 C，那么用 $\overline{K}$ 加密 $\overline{P}$ 成 $\overline{C}$。

5. (a) 令 $K=111...111$ 是由 1 组成的 DES 密钥，证明：如果 $E_K(P)=C$，那么 $E_K(C)=P$，因此用这个密钥加密两次可以得到明文。

(b) 找出在 (a) 中与 K 有同样属性的另一个密钥。

6. 假设三重 DES 通过选择两个密钥 K_1，K_2 并计算 $E_{k1}(E_{k2}(E_{k2}(m)))$ 来执行（注意密钥的顺序与通常的两密钥版本的三重加密有所不同），说明用基于中间相遇攻击方法怎样攻击这个修改后的版本。

高级加密标准：Rijndael

第 5 章

1997 年，国际标准与技术研究机构呼吁寻找 DES 的替代品。条件是：新的加密算法必须允许 128、192、256 位密钥长度，它不仅能够在 128 位输入分组上工作，而且还能够在各种不同的硬件上工作，例如，它能够应用在 8 位智能卡处理机上，也能应用在普通的 32 位个人计算机上。速度和密码强度同样也被仔细考虑过。1998 年，加密委员会对 15 种候选算法进行评定，最后选择出 5 种，分别是：MARS（来自 IBM）、RC6（来自 RSA 实验室）、Rijndael（来自 Joan Daemen 和 Vincent Rijmen）、Serpent（来自 Ross Anderson，Eli Biham 和 Lars Knudsen）和 Twofish（来自 Bruce Schneier，John Kelsey，Doug Whiting，David Wagner）。最终，Rijndael 被选为高级加密标准，余下的 4 种候选算法也很强，而且有可能在未来的加密体制中得到广泛的应用。

和其他的分组密码相比，Rijndael 能应用在多种模式下，例如 ECB、CBC 和 CFB（见 4.5 节）。

在继续讲述这个算法之前，我们先回答一个非常基本的问题，Rijndael 怎样发音？我们引用他们的网页：

如果你是荷兰人、佛兰德人、印度尼西亚人、苏里南人或者南非人，它的发音可能与你所认为的一样，否则，你的发音可以像“Reign Dahl”、“Rain Doll”或“Rhine Dahl”，我们不过分讲究，只要你让它听起来与“Region Deal”不同就可以了。

5.1 基本算法

Rijndael 算法适用于 128、192 和 256 位的密钥长度，为了简单起见，我们将限制密钥长度为 128 位，首先给出算法的简要概述，然后详细地描述算法的各个部分。

算法由 10 轮循环组成，每一轮循环都有一个循环密钥，它来自于初始密钥。这里有一个第 0 轮循环密钥，它就是初始密钥。每一轮循环输入的是 128 位，产生的输出也是 128 位。

每一循环由 4 个基本步骤组成，称之为层（layer）：

1. **字节转换（The ByteSub Transformation）**：这是一个非线性层，目的是防止微分和线性密码体制的攻击。

2. **移动行变换（The ShiftRow Transformation）**：这一步是线性组合，可以导致多轮循环各个位间的扩散。

3. **混合列变换（The MixColumn Transformation）**：与行变换的目的是相同的。

4. **加循环密钥（AddRoundKey）**：循环密钥同上层结果进行异或运算。

一个循环就是：

→字节（BS）→移动行（SR）→混合列（MC）→加循环密钥（ARK）→。

综上所述，我们归纳如下：

Rijndael 加密
1. ARK，使用第 0 个循环密钥。
2. BS，SR，MC，ARK 共循环 9 次，分别使用 1 ~9 个循环密钥。
3. 最后一个循环：BS，SR，ARK，使用第 10 个循环密钥。

最后一个循环用到了字节转换、移动行变换和加循环密钥这几步，唯独没有用到混合列变换（其原因我们在解密部分再进行解释）。

128 位的输出是一个密文分组。

5.2 层

现在我们详细地介绍具体的步骤。128 个输入位分成 16 个字节，每字节 8 位，记为：

$$a_{0,0}, a_{1,0}, a_{2,0}, a_{3,0}, a_{0,1}, a_{1,1}, \cdots, a_{3,3}。$$

将它们组成一个 4 ×4 的矩阵

$$\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix}。$$

接下来，我们需要研究一下有限域 $GF(2^8)$，在 3.10 节已经讲过。现在我们只需了解有限域的下列知识：$GF(2^8)$ 的元素是按字节计算的，一个字节 8 位，它们可进行异或加运算，也可以一种特定方式进行乘法运算（即，两字节相乘仍然是一个字节），但这个过程更复杂一些。除了字节 0 以外，每一个字节 b 都有一个乘法逆元素，即存在一字节 b'，使得 $b \cdot b' = 00000001$ 。既然我们能按字节来进行算数运算，当然也能利用元素是字节的矩阵进行运算。

作为一个技术点，我们注意到，模式 $GF(2^8)$ 依赖于一个阶数为 8 的不可约多项式的选择，Rijndael 的选择是 $X^8 + X^4 + X^3 + X + 1$ ，这就是 3.10 节例子中用到的多项式，选择其他的多项式也有可能得到同样的好算法。

5.2.1 字节转换

在这个步骤中，每个字节在矩阵中通过表 5.1 所示的 S-盒被转换为另一个字节。

写一个 8 位字节：*abcdefgh*，找到 *abcd* 行和 *efgh* 列的位置（行和列都是从 0 ~15 进行编

表 5.1 Rijndael 的 S-盒

S-盒															
99	124	119	123	242	107	111	197	48	1	103	43	254	215	171	118
202	130	201	125	250	89	71	240	173	212	162	175	156	164	114	192
183	253	147	38	54	63	247	204	52	165	229	241	113	216	49	21
4	199	35	195	24	150	5	154	7	18	128	226	235	39	178	117
9	131	44	26	27	110	90	160	82	59	214	179	41	227	47	132
83	209	0	237	32	252	177	91	106	203	190	57	74	76	88	207
208	239	170	251	67	77	51	133	69	249	2	127	80	60	159	168
81	163	64	143	146	157	56	245	188	182	218	33	16	255	243	210
205	12	19	236	95	151	68	23	196	167	126	61	100	93	25	115
96	129	79	220	34	42	144	136	70	238	184	20	222	94	11	219
224	50	58	10	73	6	36	92	194	211	172	98	145	149	228	121
231	200	55	109	141	213	78	169	108	86	244	234	101	122	174	8
186	120	37	46	28	166	180	198	232	221	116	31	75	189	139	138
112	62	181	102	72	3	245	14	97	53	87	185	134	193	29	158
225	248	152	17	105	217	142	148	155	30	135	233	206	85	40	223
140	161	137	13	191	230	66	104	65	153	45	15	176	84	187	22

号）。将这一项转换为二进制数，就是它的输出。例如，如果输入字节是 10001011，我们看到行 8（第 9 行）、列 11（第 12 列）位置的值是 61，其二进制表示为 111101，这就是 S-盒的输出。

字节转换的输出结果是一个 4×4 字节的矩阵，我们称之为：

$$\begin{pmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \end{pmatrix}。$$

5.2.2 移动行变换

矩阵的 4 行分别按偏移量 0，1，2 和 3 循环左移，得到：

$$\begin{pmatrix} c_{0,0} & c_{0,1} & c_{0,2} & c_{0,3} \\ c_{1,0} & c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,0} & c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,0} & c_{3,1} & c_{3,2} & c_{3,3} \end{pmatrix} = \begin{pmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,1} & b_{1,2} & b_{1,3} & b_{1,0} \\ b_{2,2} & b_{2,3} & b_{2,0} & b_{2,1} \\ b_{3,3} & b_{3,0} & b_{3,1} & b_{3,2} \end{pmatrix}。$$

5.2.3 混合列变换

如 3.10 节所述，有限域 $GF(2^8)$ 的一个元素被认为是一个字节，那么移动行变换的输出结果是一个 4×4 的矩阵 $(c_{i,j})$，作为 $GF(2^8)$ 的元素值，将它乘以一个矩阵，结果再作为 $GF(2^8)$ 的元素，得到输出为 $(d_{i,j})$，变换过程如下：

$$\begin{pmatrix} 00000010 & 00000011 & 00000001 & 00000001 \\ 00000001 & 00000010 & 00000011 & 00000001 \\ 00000001 & 00000001 & 00000010 & 00000011 \\ 00000011 & 00000001 & 00000001 & 00000010 \end{pmatrix} \begin{pmatrix} c_{0,0} & c_{0,1} & c_{0,2} & c_{0,3} \\ c_{1,0} & c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,0} & c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,0} & c_{3,1} & c_{3,2} & c_{3,3} \end{pmatrix}$$

$$= \begin{pmatrix} d_{0,0} & d_{0,1} & d_{0,2} & d_{0,3} \\ d_{1,0} & d_{1,1} & d_{1,2} & d_{1,3} \\ d_{2,0} & d_{2,1} & d_{2,2} & d_{2,3} \\ d_{3,0} & d_{3,1} & d_{3,2} & d_{3,3} \end{pmatrix}。$$

5.2.4 加循环密钥

循环密钥是以某种方式从密钥得来的，随后我们要介绍它，它包括 128 位，这 128 位按字节排列成一个 4×4 的矩阵（$k_{i,j}$）。将它和混合列变换的输出结果异或得：

$$\begin{pmatrix} d_{0,0} & d_{0,1} & d_{0,2} & d_{0,3} \\ d_{1,0} & d_{1,1} & d_{1,2} & d_{1,3} \\ d_{2,0} & d_{2,1} & d_{2,2} & d_{2,3} \\ d_{3,0} & d_{3,1} & d_{3,2} & d_{3,3} \end{pmatrix} \oplus \begin{pmatrix} k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} \\ k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} \\ k_{2,0} & k_{2,1} & k_{2,2} & k_{2,3} \\ k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3} \end{pmatrix} = \begin{pmatrix} e_{0,0} & e_{0,1} & e_{0,2} & e_{0,3} \\ e_{1,0} & e_{1,1} & e_{1,2} & e_{1,3} \\ e_{2,0} & e_{2,1} & e_{2,2} & e_{2,3} \\ e_{3,0} & e_{3,1} & e_{3,2} & e_{3,3} \end{pmatrix}。$$

这就是循环的最后输出结果。

5.2.5 密钥计划表

最初的密钥包括 128 位，被排成一个 4×4 的字节矩阵，这个矩阵附加 40 列后扩展如下，前 4 个列分别标记为 $W(0)$，$W(1)$，$W(2)$，$W(3)$。新的列基本上是循环产生的，假设列的增加通过已经定义的 $W(i-1)$ 列来达到，如果 i 不是 4 的倍数，那么

$$W(i) = W(i-4) \oplus W(i-1),$$

如果 i 是 4 的倍数，那么

$$W(i) = W(i-4) \oplus T(W(i-1)),$$

其中，$T(W(i-1))$ 是 $W(i-1)$ 做如下变换得到的，令列 $W(i-1)$ 的元素是 a,b,c,d，循环移位后得到 b,c,d,a。现在用字节转换步骤中 S-盒里相应的元素来取代这些字节，得到 4 字节 e,f,g,h，最后在有限域 $GF(2^8)$ 中计算循环常量：

$$r(i) = 00000010^{(i-4)/4}$$

（回想一下在这种情况下 i 是 4 的倍数），那么 $T(W(i-1))$ 就是列向量

$$(e \oplus r(i), f, g, h)。$$

从这一点来说，$W(4), ..., W(43)$ 都是由最初始的 4 列得到的。

第 i 个循环的**循环密钥**（**round key**）由以下列组成：

$$W(4i), W(4i+1), W(4i+2), W(4i+3)。$$

5.2.6 S-盒的构成

尽管 S-盒是被当作一个查询表使用的，但它有一个简单的数字描述，开始字节是 $x_7x_6x_5x_4x_3x_2x_1x_0$，x_i 是二进制位。如 3.10 节中所述，计算它在 $GF(2^8)$ 中的逆。如果字节是 00000000，它没有逆，则使用 00000000 代替它的逆，最后得到的字节 $y_7y_6y_5y_4y_3y_2y_1y_0$ 代表一个 8 维的列向量，而最右边的位 y_0 在最上方。乘以一个矩阵，并加上列向量（1，1，0，0，0，1，1，0），得到一个向量（z_0，z_1，z_2，z_3，z_4，z_5，z_6，z_7）如下：

$$\begin{pmatrix} 1&0&0&0&1&1&1&1\\ 1&1&0&0&0&1&1&1\\ 1&1&1&0&0&0&1&1\\ 1&1&1&1&0&0&0&1\\ 1&1&1&1&1&0&0&0\\ 0&1&1&1&1&1&0&0\\ 0&0&1&1&1&1&1&0\\ 0&0&0&1&1&1&1&1 \end{pmatrix}\begin{pmatrix} y_0\\ y_1\\ y_2\\ y_3\\ y_4\\ y_5\\ y_6\\ y_7 \end{pmatrix}+\begin{pmatrix} 1\\ 1\\ 0\\ 0\\ 0\\ 1\\ 1\\ 0 \end{pmatrix}=\begin{pmatrix} z_0\\ z_1\\ z_2\\ z_3\\ z_4\\ z_5\\ z_6\\ z_7 \end{pmatrix},$$

字节 $z_7z_6z_5z_4z_3z_2z_1z_0$ 是 S-盒中相应位置的项。

例如，开始字节为 11001011。它在有限域 $GF(2^8)$ 中的逆为 00000100，同 3.10 节中的计算。现在我们计算：

$$\begin{pmatrix} 1&0&0&0&1&1&1&1\\ 1&1&0&0&0&1&1&1\\ 1&1&1&0&0&0&1&1\\ 1&1&1&1&0&0&0&1\\ 1&1&1&1&1&0&0&0\\ 0&1&1&1&1&1&0&0\\ 0&0&1&1&1&1&1&0\\ 0&0&0&1&1&1&1&1 \end{pmatrix}\begin{pmatrix} 0\\ 0\\ 1\\ 0\\ 0\\ 0\\ 0\\ 0 \end{pmatrix}+\begin{pmatrix} 1\\ 1\\ 0\\ 0\\ 0\\ 1\\ 1\\ 0 \end{pmatrix}=\begin{pmatrix} 1\\ 1\\ 1\\ 1\\ 1\\ 0\\ 0\\ 0 \end{pmatrix},$$

得到字节 00011111，前 4 位 1100 表示二进制数 12，后 4 位 1011 表示二进制数 11，将每个数字减 1（因为第一行和第一列记为 0），查看 S-盒的第 11 列第 10 行，这一项是 31，用二进制表示是 00011111。

设计 S-盒时需要考虑如下因素：关系 $x \mapsto x^{-1}$ 设计成非线性。然而，简单的对应关系有可能被攻击，所以它又包括对矩阵的乘法和加向量的操作，正如前面描述的那样。矩阵的选择主要是因为它形式简单（注意，其余行是如何进行移动的），向量的选取是为了保证没有一个输入等于它对应的 S-盒的输出或者输出的补（补表示将 1 变为 0，0 变为 1）。

5.3 解　密

解密的每一步骤是加密过程中字节转换（ByteSub）、移动行（ShiftRow）、混合列

（MixColumn）和加循环密钥（AddRoundKey）的相反过程。

1. 字节转换的逆是另一种查找表，我们称之为**逆字节转换（InvByteSub）**。

2. 移动行的逆过程是用循环右移代替循环左移，得到**逆移动行（InvShiftRow）**。

3. 混合列的逆之所以存在，是因为混合列中所用的 4×4 矩阵是可逆的，**逆混合列（InvMixColumn）**变换是乘以下面这个矩阵：

$$\begin{pmatrix} 00001110 & 00001011 & 00001101 & 00001001 \\ 00001001 & 00001110 & 00001011 & 00001101 \\ 00001101 & 00001001 & 00001110 & 00001011 \\ 00001011 & 00001101 & 00001001 & 00001110 \end{pmatrix}。$$

4. 加循环密钥就是它自身的反序。

Rijndael 加密包括下列步骤：

ARK

BS，SR，MC，ARK

…

BS，SR，MC，ARK

BS，SR，ARK。

我们注意到，在最后一个循环中没有用到 MC。

为了解密，需要进行与上述步骤相反的操作。得到下面的解密过程：

ARK，ISR，IBS

ARK，IMC，ISR，IBS

…

ARK，IMC，ISR，IBS

ARK。

但是，我们还想改写这个解密过程，使它看起来更像加密。

注意到，先用 BS 后用 SR 与先用 SR 后用 BS 是完全一样的。这是由于 BS 一次作用于一个字节，SR 改变字节的排列顺序。相应地，ISR 与 IBS 的顺序也是可以颠倒的。

我们也想颠倒 ARK 和 IMC 的顺序，但这是不可能的。相反，我们进行下列操作：对一个矩阵 $(c_{i,j})$ 先应用 MC，然后应用 ARK，表示如下：

$$(c_{i,j}) \to (m_{i,j})(c_{i,j}) \to (e_{i,j}) = (m_{i,j})(c_{i,j}) \oplus (k_{i,j}),$$

其中，$(m_{i,j})$ 是混合列中用到的 4×4 矩阵，$(k_{i,j})$ 是循环密钥矩阵。$(c_{i,j})$ 关于 $(e_{i,j})$ 的逆矩阵通过计算 $(e_{i,j}) = (m_{i,j})(c_{i,j}) \oplus (k_{i,j})$ 得到，即 $(c_{i,j}) = (m_{i,j})^{-1}(e_{i,j}) \oplus (m_{i,j})^{-1}(k_{i,j})$。因此，变换的过程为：

$$(e_{i,j}) \to (m_{i,j})^{-1}(e_{i,j}) \to (m_{i,j})^{-1}(e_{i,j}) \oplus (k'_{i,j}),$$

其中，$(k'_{i,j}) = (m_{i,j})^{-1}(k_{i,j})$。第一个箭头是简单地应用逆混合列得到 $(e_{i,j})$。如果令**逆加循环密钥（InvAddRoundKey）**与 $(K'_{i,j})$ 异或，那么“先 MC 再 ARK”的逆就是“先 IMC 后 IARK”。所以，在解密的过程中我们可以用步骤“先 IMC 再 IARK”取代步骤“先 ARK 再 IMC”。

现在，我们看到解密过程改变如下：

ARK，IBS，ISR

IMC，IARK，IBS，ISR

…

IMC，IARK，IBS，ISR

ARK。

重新组织这些行，得到最终解密的步骤：

Rijndael 解密
1. ARK，使用第 10 个循环密钥。
2. IBS，ISR，IMC，IARK 共循环 9 次，分别使用 9 到 1 个循环密钥。
3. 最后一个循环：IBS，ISR，ARK，使用第 0 个循环密钥。

所以，解密本质上和加密有相同的结构，但除了第一步和最后一步，字节转换（BS）、移动行（SR）和混合列（MC）被它们的逆所替换，而加循环密钥（ARK）被逆加循环密钥（IARK）所替换。当然，循环密钥使用起来应该颠倒顺序，所以第一个加循环密钥（ARK）使用第 10 个循环的密钥，最后一个加循环密钥（ARK）使用第 0 个循环的密钥。

接下来我们解释一下为什么在最后的循环中不用混合列（MC）。假定循环中用了它，那么数据加密将从 ARK，BS，SR，MC，ARK，…开始，以 ARK，BS，SR，MC，ARK 结束，因此解密将以 IMC，IARK，IBS，ISR，… 开始，这意味着解密不必以 IMC 开始。否则，会降低算法的速度。

加密的另一种方式是，以 ARK 开始，然后有如下半循环交替序列：

（BS，SR），（MC，ARK），（BS，SR），…，（MC，ARK），（BS，SR），

最后以 ARK 结束。而解密则是，以 ARK 开始，然后有如下半循环交替序列：

（IBS，ISR），（IMC，IARK），（IBS，ISR），…，（IMC，IARK），（IBS，IBS），

最后以 ARK 结束。从中我们也可以看出，最后的 MC 不适合在任何半循环中，自然要被删去。

在 8 位处理器中，解密没有加密那么快。这是因为，在逆混合列（IMC）的 4×4 矩阵中的元素比混合列（MC）中的元素更复杂，这使解密比加密多 30% 的处理时间。然而，在许多应用中，解密是不需要的，例如，当使用 CFB 方式时（见 4.5 节），就不需要解密。因此，比较其速度的快慢并没有太大的意义。

就解密与加密在实际使用中不完全相同这一点，相比 DES 和其他一些算法来说，它没有任何弱的密钥。

5.4 设计中要考虑的问题

Rijndael 算法并不是一种 Feistel 体制（见 4.1 节和 4.2 节）。在 Feistel 体制中，每次循环时有一半的二进制位被移动，但并没有改变。在 Rijndael 中，所有的二进制数同等对待，这使得输入位扩散影响更快，它表明两轮循环就足够得到完全的扩散，即，所有的 128 个输出二进制位都完全依赖于 128 个输入二进制位。

S-盒用一种简单而清晰的方法来建造，这样就可以避免任何建立在算法上的陷门，较好

地避免存留在 DES 上的关于 S-盒的一些神秘色彩。Rijndael 的 S-盒是高度非线性的，因为它基于有限域 $GF(2^8)$ 中 $x \mapsto x^{-1}$ 的映射。用它来对付微分和线性密码分析效果显著，它还可以用来对付最新的被称为篡改（interpolation）的攻击。

移动行这一步的加入可用来对付两种最近新出现的攻击，即截短差分（truncated differential）和平方攻击（Square attack）。

混合列可达到字节扩散的目的，这步中一个输入字节的改变总是导致所有 4 个输出字节的改变。如果两个输入字节被改变，那么至少 3 个输出字节要改变。

密钥计划表涉及到了密钥位的非线性组合，因为它使用了 S-盒，这种非线性组合用来对付当解密者知道了部分密钥并以此去推测余下位的攻击。而且，它主要是为了确保两个不同密钥的大部分位都不同。循环常量是用来消除在循环过程中生成每一个循环差别的对称性。

循环的次数之所以选择 10，是因为存在蛮力攻击 6 轮循环的算法。至今还没有出现攻击 7 个或更多循环的算法。多出的 4 个循环能够让人有一种安全感。当然，循环的次数还可以增加。

RSA 算法

第 6 章

6.1 RSA 算法

艾丽斯想要给鲍勃发送一条信息，在此之前，他们没有任何联系，并且他们也未花额外的时间用密钥发送一个暗号。因此，艾丽斯发送给鲍勃的所有信息都有可能被伊芙窃走。然而，就是在这样一种信息传递方式下，仍然有办法使鲍勃能够正确读到信息而伊芙做不到。

用以前讨论过的那些办法是行不通的。艾丽斯可能发送密钥，但是伊芙有可能窃取它，那么她就有可能解密后续的所有信息。目前现有的体制中，有一种称为**公开密钥密码体制**(**public key cryptosystem**)，最初是由 Diffie 和 Hellman 在他们的论文［Diffie-Hellman］中提出的。但是，他们并没有在实际中应用（尽管他们确实提出了另外一种密钥交换程序，这种程序能够工作在公共通道上；见 13.1 节）。在以后的几年里，又有许多方法被提出。其中最成功的就是众所周知的 RSA 算法，是由 Rivest，Shamir 和 Adleman 于 1977 年提出的，这种方法基于把一个整数因数分解为几个素数是非常困难的这种观点。

长期以来有人声称，政府加密机构早在许多年之前就发现了 RSA 算法，但是为了保密起见，一直未将此泄漏出去。最后，在 1997 年，从英国的一个加密机构 CESG 发布的文件中得知，James Ellis 在 1970 年就已经发现了公开密钥密码体制，随后，在 1973 年，Clifford Cocks 写下了一份内部文件，描述了 RSA 算法，其中的加密指数 e（见随后的讨论）与模 n 是完全相同的。

下面是 RSA 算法的工作过程。鲍勃选择了两个不相同的大素数 p 和 q，将它们相乘得到：

$$n = pq。$$

他又选择了一个加密指数 e，使得：

$$\gcd(e,(p-1)(q-1)) = 1。$$

他将一对 (n,e) 发送给艾丽斯，但却不告知 p 和 q 的值。并且，艾丽斯也很有可能是鲍勃的对手，她不需要知道 p 和 q 去把她的信息安全地发送给鲍勃。艾丽斯把她的信息编为数字 m。如果 m 比 n 大，她就把信息分成组，其中每一组都小于等于 n。为了简便起见，假设 $m > n$，艾丽斯计算：

$$c \equiv m^e \pmod{n}$$

并将 c 发送给鲍勃，由于鲍勃知道 p 和 q 的值，他可以计算 $(p-1)(q-1)$，然后求出解密指

数 d. 利用：

$$de \equiv 1(\bmod (p-1)(q-1))。$$

接着，我们会发现，

$$m \equiv c^d (\bmod n),$$

这样鲍勃就可以读懂信息了。

我们将这个算法归纳为下表：

RSA 算法

1. 鲍勃选择保密的素数 p 和 q，并计算 $n = pq$；
2. 鲍勃通过 $\gcd(e,(p-1)(q-1)) = 1$ 来选择 e；
3. 鲍勃通过 $de \equiv 1(\bmod (p-1)(q-1))$ 来计算 d；
4. 鲍勃将 n 和 e 设为公开的，而将 p，q，d 设为秘密的；
5. 艾丽斯将 m 加密为 $c \equiv m^e (\bmod n)$，并将 c 发送给鲍勃；
6. 鲍勃通过计算 $m \equiv c^d (\bmod n)$ 解密。

例如：鲍勃选择

$$p = 885320963, q = 238855417,$$

那么，

$$n = p \cdot q = 211463707796206571,$$

设加密系数为

$$e = 9007,$$

将 n 和 e 的值发送给艾丽斯。

艾丽斯传递的信息是 *cat*。由于在前面章节中我们知道字母 $a=0$；现在用 $a=01$ 直至 $z=26$来代替前面的字母表示。根据前面章节中所讲述的方法，如果字母 a 出现在信息的开头，那么信息 m 将以 00 开始，因此，字母 a 这个信息将不存在。

于是信息变为

$$m = 30120。$$

艾丽斯计算

$$c \equiv m^e \equiv 30120^{9007} \equiv 113535859035722866 \ (\bmod n),$$

她将 c 传送给鲍勃。

由于鲍勃知道 p 和 q 的值，所以他能够计算 $(p-1)(q-1)$，他用扩展的欧几里得算法（见 3.2 节）计算 d，使得

$$de \equiv 1 \ (\bmod (p-1)(q-1)),$$

得：

$$\mathrm{d} = 116402471153538991,$$

然后鲍勃计算

$$c^d \equiv 113535859035722866^{116402471153538991} \equiv 30120 \ (\bmod n),$$

因此他可以得到最初的信息。 ■

以上有几点需要解释一下，但最重要的是为什么 $m \equiv c^d \pmod n$。回顾欧拉定理（见3.6节）：如果 $\gcd(a,n)=1$，那么 $a^{\phi(n)} \equiv 1 \pmod n$。在上述事例中，$\phi(n)=\phi(pq)=(p-1)(q-1)$。假设 $\gcd(m,n)=1$，这是非常可能的；因为 p 和 q 是很大的，m 不能够作为一个因子，又因为 $de \equiv 1 \pmod{\phi(n)}$，我们可以记 $de = 1 + k\phi(n)$，k 是一个整数。所以，

$$c^d \equiv (m^e)^d \equiv m^{1+k\phi(n)} \equiv m \cdot (m^{\phi(n)})^k \equiv m \cdot 1^k \equiv m \pmod n。$$

以上我们证实了鲍勃可以恢复信息。如果 $\gcd(m,n) \neq 1$，鲍勃仍然可以恢复信息，见练习13。

那么伊芙是如何窃取信息的呢？她截获 n，e，c，但是并不知道 p，q，d。我们假设伊芙不可能将 n 因数分解。计算 d 显然需要知道 $\phi(n)$，后面我们会证明它等价于知道 p 和 q。还有没有其他的方法呢？现在我们证明：如果伊芙能够找到 d，就能够将 n 因数分解，因此伊芙不可能找到 d。

由于伊芙知道 $c \equiv m^e \pmod n$，为什么不能简单地计算出 c 的 e 次方根呢？这是因为工作在模 n 的方式下计算这种情况是很困难的。比如，已知 $m^3 \equiv 3 \pmod{85}$，你并不能计算出3的立方根，即1.2599...，然后推测出模85的结果。当然若用一对一的搜索，最终可以得到 $m=7$，但这个方法对大的 n 值是不可行的。

鲍勃怎样来选择 p 和 q 呢？原则上讲，它们彼此应该是随机的和独立的，长度多少取决于安全的需要，但至少应该有100位，具体原因后面会讲到，最好它们的长度稍微有所不同。当讨论到素数判定的时候，会发现这样的素数生成起来相当快，对 p 和 q 的一些其他的试验也证明是不错的。例如，如果 $p-1$ 有一个小的素数因子，那么 n 是很容易通过 $p-1$ 的方法（见6.4节）分解的，因此 p 可以忽略或用其他的素数取代。

为什么鲍勃需要条件 $\gcd(e,(p-1)(q-1))=1$ 呢？回顾一下（见3.3节），$de \equiv 1 \pmod{(p-1)(q-1)}$ 有一个解 d，当且仅当 $\gcd(e,(p-1)(q-1))=1$。因此，为了保证 d 是存在的，需要这个条件。用扩展的欧几里得算法可以很快地求出 d。由于 $p-1$ 是偶数，$e=2$ 是不可能的，然后，尝试 $e=3$。然而，如果 e 比较小的话，就有许多不利因素（见6.2节和上机题14），因此建议使用大一点的 e 值。例如：可以设 e 为一个中等大小的素数，这样就不难保证 $\gcd(e,(p-1)(q-1))=1$ 了。

在加密过程中，艾丽斯需要计算 $m^e \pmod n$，回忆一下其计算速度相当快，且不耗费大量内存，如逐项平方。这无疑是模运算的一个优点：如果艾丽斯想要先计算 m^e，然后再计算模 n，这样 m^e 很有可能内存溢出。同样，在解密过程中也可以快速有效地计算出 $c^d \pmod n$。因此，加密和解密过程中的所有计算都能够快速进行完毕（时间大约是 $\log n$ 的幂次方）。通过假设 n 不能够被因数分解来保证安全性。

我们做了两个假设，在这儿来证明它们。这两个假设主要说明的是，求 $\phi(n)$ 和解密指数 d 的难度与因数分解 n 的难度是一样的。因此，如果将一个数因数分解十分困难，那么也就没有快速求解 d 的捷径。

假设1：假设 $n=pq$ 是两个不同素数的乘积，如果已知 n 和 $\phi(n)$，那么就能够很快地求出 p 和 q。

注意，

$$n - \phi(n) + 1 = pq - (p-1)(q-1) + 1 = p + q。$$

因此，我们知道了 pq 和 $p+q$。二次多项式

$$X^2-(n-\phi(n)+1)X+n=X^2-(p+q)X+pq=(X-p)(X-q)$$

的根为 p 和 q，也可以根据二项式的求根公式计算：

$$p,q=\frac{(n-\phi(n)+1)\pm\sqrt{(n-\phi(n)+1)^2-4n}}{2},$$

得出 p 和 q。

例如，假设 $n=221$，已知 $\phi(n)=192$，二次多项式

$$X^2-30X+221$$

的根为：

$$p,q=\frac{30\pm\sqrt{30^2-4\cdot 221}}{2}=13,17。$$

假设 2：如果已知 d 和 e，就能够将 n 因数分解。

在 6.4 节关于因数分解方法的讨论中，证明了如果存在一个一般指数 $b>0$，使得对于所有的 a，满足 $\gcd(a,n)=1$，都有 $a^b\equiv 1(\mathrm{mod}\ n)$，那么就能够将 n 因数分解。因为 $de-1$ 是 $\phi(n)$ 的倍数，记 $de-1=k\phi(n)$，可以得到：

$$a^{de-1}\equiv(a^{\phi(n)})^k\equiv 1(\mathrm{mod}\ n)$$

当 $\gcd(a,n)=1$ 时，这种方法对一般的指数都适用。

RSA 算法的一种应用是，例如，有几家银行需要互相发送财政数据，如果有上千家银行，那么在每两家银行之间都有一个密钥来秘密通信是不可行的。但可以用下述的这种方法，每一家银行事先选取两个整数 n 和 e，然后将它们出版在一本公开的书中。假设银行 A 想传送数据给银行 B，那么 A 查寻 B 的 n 和 e，并用它们传送信息。实际上，RSA 算法在传送大量数据时，并非足够快。因此 RSA 算法经常被用来传送快速加密算法的密钥，如 DES。

6.2 对 RSA 的攻击

在实际中，只要 RSA 算法执行正确，就被证明是很高效的，在练习中我们给出一些可能的执行错误，这里还有一些其他的潜在困难。注意在结果的各项中，我们要小心地选择 d，一种方法是先选择 d，然后用 $de\equiv 1(\mathrm{mod}\ \phi(n))$ 找出 e。

定理：令 $n=qp$，p 和 q 是满足 $q<p<2q$ 的素数。假设 $d<\frac{1}{3}n^{1/4}$，给定 (n,e) 满足 $de\equiv 1(\mathrm{mod}\phi(n))$，这是一个计算 d 的很有效的过程。

这种方法使用了 e/n 的连分数，在［Wiener］中有介绍。如果 p 和 q 的长度有稍微的差别，且 d 要大一些，这样的结果必然有好的安全性。但是，这也有不好的一面，因为小的 d 值能使解密速度更快。

定理：令 $n=pq$ 有 m 位，如果我们知道 p 的前 $m/4$ 位或者是后 $m/4$ 位，那么就能够将 n 因数分解。

换言之，如果 p 和 q 有 100 位，我们知道 p 的前 50 位或后 50 位，那么我们就能将 n 因数分解。因此，如果从一个随机的开始点来选择素数 p，这种方法应该保证大量的 p 值是不可预测的。例如，假设我们随机地选取一个 50 位的数 N 并测试 $N\cdot 10^{50}+k, k=1,3,5,\ldots$，从头开始直到找到素数 p（它可能发生在 $k<1000$ 的情况下）。使用该方法的攻击者将会知

道最后 50 位数的 47 位（除了最后 3 位它们可能全为 0），对 $k < 1000$ 的各种值，尝试定理所述的方法最终将会得到 n 的因数分解。

关于前面叙述结果的细节，详见［Coppersmith2］。相关的结论如下。

定理：假设 (n,e) 是一个 RSA 的公开密钥，n 有 m 位数，令 d 是解密指数，如果我们至少拥有 d 的最后 $m/4$ 位，那么我们就能够及时有效地找到 d，花费的时间是线性函数 $e\log_2 e$。

这表明找到 d 的时间和线性函数 $e\log_2 e$ 是相关的，如果 e 很小，那么当我们知道大部分 d 的时候我们就能很快地找到 d；如果 e 很大，接近于 n，那么定理不比一个一个搜索 d 来得快，详见［Boneh et al.］。

更多关于攻击 RSA 的例子见［Boneh］。

定时攻击

另一种对 RSA 及类似体制的攻击类型是 Paul Kocher 在 1995 年发现的，当时他还是斯坦福的本科生，他指出通过仔细定时地计算一系列解密的时间是有可能发现解密指数的。尽管有阻止进攻的方法，这样的发展还是让人不安，因为随着对数学理解得日益深刻，对于安全的感觉也开始全面，Kocher 的攻击证明一个体制仍然存在许多难以预知的弱点，

现在来解释定时攻击是怎么工作的，假设伊芙能够观察到鲍勃解码时的一些密文 y，她测出取出每个 y 所用的时间，知道了 y 和译码时调用它的时间，她就能够找到解密指数 d。但是，伊芙最开始是怎样得到这些信息的呢？在加密信息传送给鲍勃，鲍勃的电脑自动解密和回应的时候，测量响应的时间间隔足够达到目前的目的。

假设我们知道硬件在计算 $y^d \pmod n$ 时的工作原理，我们可以利用这些信息测出计算时间，为以后可能会发生的情况做好准备，

假设 $y^d \pmod n$ 被练习 3.12 的算法算出，具体过程如下。

设 $d = b_1 b_2 ... b_w$ 被转换为二进制（例如，当 $x = 1011$ 时，有 $b_1 = 1, b_2 = 0, b_3 = 1, b_4 = 1$），设 y 和 n 是整数，执行下列过程：

1. 初始值 $k = 1, s_0 = 1$；
2. 如果 $b_k = 1$，设 $r_k \equiv s_k y \pmod n$。如果 $b_k = 0$，设 $r_k = s_k$；
3. 设 $s_{k+1} \equiv r_k^2 \pmod n$；
4. 如果 $k = w$，终止退出，如果 $k < w$，k 加 1 转到步骤 2。

最后 $r_w \equiv y^d \pmod n$。

注意到仅当 $b_k = 1$ 时才与 $s_k y$ 相乘，在许多情况下，乘法运算有一个合理的区间，假设本例是这种情况。

在我们继续之前，需要知道，假设有一个随机过程输出的是实数 t。对我们来说，如果随机输入一个 y 值，t 就对应着计算机完成计算所花费的时间，表示是这些输出的平均值。如果我们记录输出 $t_1, ..., t_n$，它们的平均值近似为 $m = (t_1 + \cdots t_n)/n$，随机过程的方差可以近似为：

$$\mathrm{Var}(\{t_i\}) = \frac{(t_1 - m)^2 + \cdots + (t_n - m)^2}{n}。$$

标准偏移量就是方差的平方根与给定的方差（在这里就是 t_i 的值）的偏差。

我们需要的一个重要的事实是：两个随机过程是相互独立的，它们输出和的方差等于两个过程方差的和。例如，计算机将一个计算分解为两个独立的过程，需要的时间分别是 t' 和

t''。总时间 t 为 $t'+t''$，因此，$\mathrm{Var}(\{t_i\})$ 大致为 $\mathrm{Var}(\{t'_i\})+\mathrm{Var}(\{t''_i\})$。

现在假设伊芙知道密文 $y_1, \ldots, y_n$ 和计算每个 $y_i^d \pmod n$ 的时间，假设她知道解密指数 d 的位 $b_1, \ldots, b_{k-1}$，因为她知道硬件的工作原理，所以她知道用前面的算法计算 $r_1, \ldots, r_{k-1}$ 所需要的时间，于是她知道对每个 y_i，时间 t_i 就是计算 $r_k, \ldots, r_w$ 的时间。

伊芙想要确定 b_k，如果 $b_k=1$，对每个产生的密文 y_i 执行乘法 $s_k y \pmod n$，如果 $b_k=0$，不执行乘法。

设 t'_i 是计算机执行乘法 $s_k y \pmod n$ 的时间总和，尽管伊芙不知道乘法是否真的执行。设 $t''_i=t_i-t'_i$，伊芙计算出 $\mathrm{Var}(\{t_i\})$ 和 $\mathrm{Var}(\{t''_i\})$，如果 $\mathrm{Var}(\{t_i\})>\mathrm{Var}(\{t''_i\})$，那么她得出 $b_k=1$，否则 $b_k=0$。在确定 b_k 后，她可以用同样的方法找到所有的位。

为什么要这样做呢？如果乘法执行了，t'_i 就是计算机完成所有乘法的计算所用的时间，那么可以假设 t'_i 和 t''_i 的输出是彼此独立的。因此，

$$\mathrm{Var}(\{t_i\}) \approx \mathrm{Var}(\{t'_i\})+\mathrm{Var}(\{t''_i\}) > \mathrm{Var}(\{t''_i\})。$$

如果乘法没有执行，t'_i 是执行与计算无关的操作所用的时间，因此可以假设 t_i 和 t'_i 是相互独立的，于是，

$$\mathrm{Var}(\{t''_i\}) \approx \mathrm{Var}(\{t_i\})+\mathrm{Var}(\{-t'_i\}) > \mathrm{Var}(\{t_i\})。$$

注意我们不能用平均值来代替方差，因为 $\{-t_i\}$ 的平均值是负数，所以最后的不等式不成立。能从平均值中推出的是 d 中所有非零位的总数。

前面给出了这种方法的一个相对简单的版本。实际上，针对特殊的情况，需要进行各种修改，但是大体策略保持不变。要了解更多细节，请参考 [Kocher]。

6.3 素数判定

假设我们想判定一个 200 位的整数是否是素数。为什么不用所有小于它的平方根的素数来除它呢？大约有 4×10^{97} 个小于 10^{100} 的素数，这比宇宙中微粒的总数还要多得多。另外，如果计算机每秒能够处理 10^9 个素数，那么这个计算也要花费将近 10^{81} 年。很明显，需要更好的方法。本节将讨论这些方法。

许多因数分解方法中有一个非常基本的概念，如下所示。

基本原理： 令 n 表示一个整数，假设存在整数 x 和 y 满足 $x^2\equiv y^2 \pmod n$，但 $x\not\equiv \pm y \pmod n$，那么 n 是合数。而且，$\gcd(x-y, n)$ 给出了 n 的一个非平凡因子。

证明：令 $d=\gcd(x-y, n)$。如果 $d=n$，那么假定 $x\not\equiv y \pmod n$ 不会发生。假设 $d=1$，结合关于整除性的一个基本结论：如果 $a \mid bc$，并且 $\gcd(a,b)=1$，那么 $a\mid c$（参考练习 3.3）。那么在我们的例子中，既然 n 能整除 $x^2-y^2=(x-y)(x+y)$，并且 $d=1$，所以 n 必定能整除 $x+y$，这同 $x\not\equiv -y \pmod n$ 的假设是矛盾的。因此，$d\neq 1,n$，所以 d 为 n 的一个非平凡因子。 □

例： 因为 $12^2\equiv 2^2 \pmod{35}$，但是 $12\not\equiv \pm 2 \pmod{35}$。我们知道 35 是一个合数。进而，$\gcd(12-2,35)=5$ 是 35 的一个非平凡因子。 ■

可能有些令人吃惊，但因数分解与素数判定是不同的。证明一个数是合数比对它因数分解要容易得多。存在许多已知为合数的大整数还没有被因数分解。这是怎样发生的呢？我们给一个简单的例子。通过费尔马定理我们知道，如果 p 是素数，那么 $2^{p-1}\equiv 1 \pmod p$。让

我们运用这个定理来证明 35 不是素数。通过连续的自乘，我们发现（同余于模 35）

$$2^4 \equiv 16,$$
$$2^8 \equiv 256 \equiv 11,$$
$$2^{16} \equiv 121 \equiv 16,$$
$$2^{32} \equiv 256 \equiv 11。$$

因此，

$$2^{34} \equiv 2^{32}2^2 \equiv 11 \cdot 4 \equiv 9 \not\equiv 1 \pmod{35}。$$

费尔马定理认为 35 不可能是素数，所以我们已经证明 35 是一个没有发现其因子的合数。

这个方法概括如下。

Miller-Rabin 素数判定。令 $n > 1$ 表示一个奇整数。令 $n-1 = 2^k m$，其中 m 为奇数。选择一个满足 $1 < a < n-1$ 的随机整数 a，计算 $b_0 \equiv a^m \pmod{n}$。如果 $b_0 \equiv \pm 1 \pmod{n}$，那么停下来，认定 n 可能为素数。否则，令 $b_1 \equiv b_0^2 \pmod{n}$。如果 $b_1 \equiv 1 \pmod{n}$，那么 n 为合数（并且 $\gcd(b_0-1, n)$ 给出了 n 的一个非平凡因子）。如果 $b_1 \equiv -1 \pmod{n}$，那么停下来，认定 n 可能为素数。否则，令 $b_2 \equiv b_1^2 \pmod{n}$。如果 $b_2 \equiv 1 \pmod{n}$，那么 n 为合数。如果 $b_2 \equiv -1 \pmod{n}$，那么停下来，认定 n 可能为素数。采用这种方法继续下去，直到停下来或达到 b_{k-1}。如果 $b_{k-1} \not\equiv -1 \pmod{n}$，那么 n 为合数。

例：令 $n = 561$，那么 $n-1 = 560 = 16 \cdot 35$，所以 $2^k = 2^4$，且 $m = 35$。令 $a = 2$，那么

$$b_0 \equiv 2^{35} \equiv 263 \pmod{561},$$
$$b_1 \equiv b_0^2 \equiv 166 \pmod{561},$$
$$b_2 \equiv b_1^2 \equiv 67 \pmod{561},$$
$$b_3 \equiv b_2^2 \equiv 1 \pmod{561}。$$

既然 $b_3 \equiv 1 \pmod{561}$，我们断定 561 为合数。另外，$\gcd(b_2 - 1, 561) = 33$，这是 561 的一个非平凡因子。 ■

如果 n 为合数，且 $a^{n-1} \equiv 1 \pmod{n}$，那么我们认为 n 是一个基数为 a 的伪素数。如果 a 和 n 是这样的，且 n 通过了 Miller-Rabin 测试，那么就认为 n 是一个基数为 a 的绝对伪素数。我们在 3.6 节中证明了 $2^{560} \equiv 1 \pmod{561}$，所以 561 是基数为 2 的伪素数。然而，前面的计算证明 561 不是一个基数为 2 的绝对伪素数。对于一个给定的基数，绝对伪素数比普通伪素数更加稀少。

直到 10^{10} 为止，共有 455052511 个素数。有 14884 个底数为 2 的伪素数和 3291 个基为 2 的绝对伪素数。因此，计算 $2^{n-1} \pmod{n}$ 将不能判定出这个范围内的一个合数，其概率小于 1/30000，并且用满足 $a = 2$ 的 Miller-Rabin 测试也有小于 1/100000 的概率会失败。

能够证明，对于随机选择 a 的 Miller-Rabin 测试判定合数失败的概率最大为 1/4。事实上，它失败的概率要比这个小得多。参考［Damgard et al］。如果随机选择 a 的值重复 10 次这个测试，那么预计把一个合数当成素数的概率最大为 $(1/4)^{10} \approx 10^{-6}$。实际上，对于单独的 a 用这个测试是十分精确的。

虽然绝对伪素数很稀少，但已经被证实，对于任意的有限基数集合 B，对所有的 $b \in B$ 有无限多个整数是绝对伪素数。对所有的基数 $b = 2,3,5,7$，第一个绝对伪素数为

3215031751。存在一个 337 位的数字，对于素数小于 200 的所有基数是绝对伪素数。

假设我们需要找到一个大约 100 位的素数。素数定理认为 x 附近素数的密度近似为 $1/\ln x$。当 $x = 10^{100}$，可以得到其密度大约为 $1/\ln(10^{100}) = 1/230$，因为能够跳过偶数，所以就可达到 1/115。挑选一个随机的起点，扔掉偶数（和另外一些小素数的倍数）。用 Miller-Rabin 测试接连地判定每一个保留下来的数字。这将去除所有的合数，平均起来，找到一个近似的素数候选数将花费不到 100 次的 Miller-Rabin 测试，所以这可以非常迅速地完成。如果我们需要完全确定被怀疑的数字是素数，还有许多更精确的素数测试方法能够在几秒钟内判定一个 100 位的数字。

这个测试为什么有效呢？例如：假设 $b_3 \equiv 1 \pmod n$，这意味着 $b_2^2 \equiv 1^2 \pmod n$，应用前面提到的基本法则，得出或者 $b_2 \equiv \pm 1 \pmod n$，或者 $b_2 \not\equiv \pm 1 \pmod n$，且 n 为合数。在后边的例子中，$\gcd(b_2 - 1, n)$ 给出了 n 的一个非平凡因子。在前边的例子中，通过前面的步骤，算法已经停下来了。如果我们达到 b_{k-1}，那么已经计算了 $b_{k-1} \equiv a^{(n-1)/2} \pmod n$。它的平方为 a^{n-1}，由费尔马定理可知，如果 n 为素数，那么 a^{n-1} 必为 $1 \pmod n$。因此，如果 n 为素数，那么 $b_{k-1} \equiv \pm 1 \pmod n$，所有其他的选择意味着 n 是合数。另外，如果 $b_{k-1} \equiv 1$，那么如果我们不在一个较早的步骤停下来，就会有 $b_{k-2}^2 \equiv 1^2 \pmod n$ 满足 $b_{k-2} \not\equiv \pm 1 \pmod n$，这表明 n 为合数（并且能够因数分解 n）。

实际上，如果 n 为合数，那么通常我们达到 b_{k-1}，并且它不等于 $\pm 1 \pmod n$。事实上，通常 $a^{n-1} \not\equiv 1 \pmod n$，这意味着费尔马定理失败了，所以 n 不是素数。

例如，令 $n = 299$，$a = 2$。因为 $2^{298} \equiv 140 \pmod{299}$，所以费尔马定理和 Miller-Rabin 测试都认为 299 不是素数（不能因数分解它）。发生的原因如下所述。注意 $299 = 13 \times 23$。一个简单的计算证明 $2^{12} \equiv 1 \pmod{13}$ 且没有更小的有效指数。事实上，当且仅当 j 是 12 的倍数时，$2^j \equiv 1 \pmod{13}$，因为 298 不是 12 的倍数，所以 $2^{298} \not\equiv 1 \pmod{13}$，而且 $2^{298} \not\equiv 1 \pmod{299}$。同样，当且仅当 j 是 11 的倍数时，$2^j \equiv 1 \pmod{23}$，由此我们可以再一次推导出 $2^{298} \not\equiv 1 \pmod{299}$。如果在这个例子中费尔马定理（和 Miller-Rabin 测试）给我们错误的答案，那么我们就应该要求 $13 \cdot 23 - 1$ 为 $12 \cdot 11$ 的倍数。

考虑 $n = pq$ 为两个素数的乘积的一般情况。为了简单起见，考虑 $p > q$ 的情况，并假设当且仅当 $k \equiv 0 \pmod{p-1}$ 时 $a^k \equiv 1 \pmod p$，这意味着 a 是模 p 的一个本原根，存在 $\phi(p-1)$ 满足 a 模 p。因为 $0 < q - 1 < p - 1$，所以可得到

$$n - 1 \equiv pq - 1 \equiv q(p-1) + q - 1 \not\equiv 0 \pmod{p-1}。$$

因此，对于我们选择的 a，$a^{n-1} \not\equiv 1 \pmod p$，这暗示着 $a^{n-1} \not\equiv 1 \pmod n$。同样的原因显示，对于 a 的许多其他的选择通常也存在 $a^{n-1} \not\equiv 1 \pmod n$。

但是假设我们在 $a^{n-1} \equiv 1 \pmod n$ 的情况下，将会发生什么呢？让我们看一下 $n = 561$ 的例子，因为 $561 = 3 \times 11 \times 17$，考虑一下序列 $b_0, b_1, b_2, b_3 \bmod 3, \bmod 11, \bmod 17$ 将会发生什么：

$$\begin{aligned}
b_0 &\equiv -1 \pmod 3), & &\equiv -1 \pmod{11}, & &\equiv 2 \pmod{17} \\
b_1 &\equiv 1 \pmod 3, & &\equiv 1 \pmod{11}, & &\equiv 4 \pmod{17} \\
b_2 &\equiv 1 \pmod 3, & &\equiv 1 \pmod{11}, & &\equiv -1 \pmod{17} \\
b_3 &\equiv 1 \pmod 3, & &\equiv 1 \pmod{11}, & &\equiv 1 \pmod{17}。
\end{aligned}$$

因为 $b_3 \equiv 1 \pmod{561}$，所以可得 $b_2^2 \equiv b_3 \equiv 1$ 模所有 3 个素数。但是 b_3 不可能是我们第一次得

到的满足 $b_i\equiv1$ 模一个特别的素数。我们已经得到 $b_1\equiv1\bmod3$ 和 mod 11，但是必须等到 b_3 对 17 取模。因此，$b_2^2\equiv b_3\equiv1\bmod3,\bmod11,\bmod17$，但 b_2 仅仅模 3 和模 11 时同余 1，因此，b_2-1 包含因子 3 和 11，但不包括 17。这就是为什么 $\gcd(b_2-1,561)$ 找到了 561 的因子 33。我们能够用这种方法因数分解 561 的原因在于序列 b_0，b_1，... 不在同一时间达到 1 对素数取模。

更一般地，考虑 $n=pq$（几个素数的乘积与前面类似）的情况，并假设 $a^{n-1}\equiv1\pmod n$。正如前面指出的，这种情况是不太可能的；但如果确实发生了，那么看一下模 p 和模 q 正在发生什么。很可能序列 $b_i\pmod p$ 和 $b_i\pmod q$ 在不同时间达到 -1 和 1，就像在 561 的例子中那样。在本例中，对于某个 i 将有 $b_i\equiv-1\pmod p$ 但 $b_i\equiv1\pmod q$；因此，$b_i^2\equiv1\pmod n$ 但 $b_i\not\equiv\pm1\pmod n$，所以我们能够因数分解 n。

n 能够通过 Miller-Rabin 测试的惟一方法是 $a^{n-1}\equiv1\pmod n$ 并且序列 $b_i\pmod p$ 和 $b_i\pmod q$ 同时达到 1，这很少发生。

要了解更多的关于素数测试及其历史的信息，请参考［Williams］。

6.4 因数分解

现在我们转向因数分解。对大多数场合来说用所有小于$\sqrt{n}$的素数 p 来整除整数 n 的基本方法太慢了，多年来，人们一直致力于研究更有效的运算法则，这里我们介绍其中的一些。在第 15 章，我们还将介绍一种用椭圆曲线的方法，在第 17 章将说明一下如果制造出量子计算机，它会怎样有效地来进行因数分解。

有一种非常慢的方法，它通常被称作**费尔马因数分解（Fermat factorization）**方法。它的思想是用两个平方的差值来表示 n：$n=x^2-y^2$，那么 $n=(x+y)(x-y)$ 给出了 n 的因数分解。例如，假设我们想因数分解 $n=295927$，计算 $n+1^2,n+2^2,n+3^2,\ldots$，直到我们找到一个平方。在本例中，$295927+3^2=295936=544^2$。因此，

$$295927=(544+3)(544-3)=547\cdot541\text{。}$$

当 n 是两个彼此非常接近的素数的乘积时，费尔马方法非常有效。如果 $n=pq$，那么将花费 $|p-q|/2$ 步才能找到它的因数分解，但如果 p 和 q 是两个随机选择的 100 位的素数，那么很可能 $|p-q|$ 非常大，可能也接近 100 位。所以，费尔马因数分解法不可能有效，虽然这样，仅仅从安全角度考虑，作为 RSA 系数的素数经常被选为稍微不同的大小。

现在我们转向更多近代的方法。表面上，Miller-Rabin 测试看起来可以很容易因数分解 n；但经常发生的情况是还没有满足 $b_u\equiv\pm1\pmod n$ 就达到了 b_{k-1}。另一方面，假设存在某个指数 r，可能不是 $n-1$，对于所有满足 $\gcd(a,n)=1$ 的 a 满足 $a^r\equiv1\pmod n$，那么经常可能因数分解 n。我们注意到这样一个指数 r 一定是偶数（如果 $n>2$）；因为我们能够获得 $a\equiv-1\pmod n$，所以需要 $(-1)^r\equiv1$。

通用指数因数分解法（Universal Exponent Factorization Method）。假设有一个指数 $r>0$，对于所有满足 $\gcd(a,n)=1$ 的 a 满足 $a^r\equiv1\pmod n$，令 $r=2^km$，其中 m 为奇数，选择一个满足 $1<a<n-1$ 的随机数 a，如果 $\gcd(a,n)\neq1$，那么可以得到 n 的一个因子，所以假定 $\gcd(a,n)=1$。令 $b_0\equiv a^m\pmod n$，并对于 $0\leqslant u\leqslant k-1$ 连续地定义 $b_{u+1}\equiv$

$b_u^2 \pmod n$。如果 $b_0 \equiv 1 \pmod n$，那么停下来，尝试用一个不同的 a，如果对于某个 u 有 $b_u \equiv -1 \pmod n$，那么停下来，尝试用一个不同的 a，如果对于某个 u 有 $b_{u+1} \equiv 1 \pmod n$ 但 $b_u \not\equiv \pm 1 \pmod n$，那么 $\gcd(b_u - 1, n)$ 给出了 n 的一个非平凡因子。

这看起来与 Miller-Rabin 测试非常相似，不同之处在于 r 的存在保证了对于某个 u 存在 $b_{u+1} \equiv 1 \pmod n$，这在 Miller-Rabin 情况下并不经常发生。尝试 a 的几个值有很大的可能可以因数分解 n。

当然，我们也许会问怎样才能找到一个指数 r。通常，这似乎很难，并且这个测试在实践中不能被应用，然而，它在证明 RSA 运算法则中，知道解密指数就可因数分解模数时是有用的。

在某些情况下，我们不知道一个通用指数，但知道指数 r 对 a 的一个值是有效的，有时这允许我们因数分解 n。

指数因数分解法（Exponent Factorization Method）。假设有一个指数 $r > 0$ 和满足 $a^r \equiv 1 \pmod n$ 的整数 a，令 $r = 2^k m$，其中 m 为奇数，令 $b_0 \equiv a^m \pmod n$，并对于 $0 \leq u \leq k-1$ 连续地定义 $b_{u+1} \equiv b_u^2 \pmod n$。如果 $b_0 \equiv 1 \pmod n$，那么停下来，这个过程因数分解 n 失败；如果对于某个 u 有 $b_u \equiv -1 \pmod n$，那么停下来，这个过程因数分解 n 失败；如果对于某个 u 有 $b_{u+1} \equiv 1 \pmod n$ 但 $b_u \not\equiv \pm 1 \pmod n$，那么 $\gcd(b_u - 1, n)$ 给出了 n 的一个非平凡因子。

当然，如果我们接受 $a=1$，那么任何 r 都是有效的。但是那样 $b_0 = 1$，所以这个方法失败了。但如果通过某种相当明智的方法找到 a 和 r，那么这种方法因数分解 n 有很大的可能。

如果 n 的素数因子之一有一个特别的属性，那么有时很容易因数分解 n。例如，如果 p 能整除 n 且 $p-1$ 仅有小素数因子，那么下面的方法是有效的，它是 1974 年 Pollard 发明的。

$p-1$ 因数分解法（The $p-1$ *Factoring Algorithm*）。选择一个整数 $a > 1$，经常用 $a = 2$。选一个上限 B，像下面这样计算 $b \equiv a^{B!} \pmod n$。令 $b_1 \equiv a \pmod n$，$b_j \equiv b_{j-1}^j \pmod n$，那么 $b_B \equiv b \pmod n$。令 $d = \gcd(b-1, n)$，如果 $1 < d < n$，那么我们就得到了 n 的一个非平凡因子。

假设 p 是 n 的一个素数因子，且 $p-1$ 只有小素数因子。那么 $p-1$ 整除 $B!$ 是可能的，也就是说 $B! = (p-1)k$。通过费尔马定理得到 $b \equiv a^{B!} \equiv (a^{p-1})^k \equiv 1 \pmod p$，所以 p 将出现在 $b-1$ 和 n 的最大公约数中。如果 q 是 n 的另外一个素数因子，那么 $b \equiv 1 \pmod q$ 是不可能的，除非 $q-1$ 也仅仅有小素数因子。如果 $d = n$，那么并非所有的都没用。在本例中，有指数 r（也就是 $B!$）和满足 $a^r \equiv 1 \pmod n$ 的 a，很有可能上面的指数因数分解法将因数分解 n。作为替代方法，我们可以选择一个较小的 B 值并重复这个计算（这与指数因数分解法所做的有些相似）。

我们怎样选择上限 B 的值呢？如果我们选择一个小 B，那么算法将快速运行，但成功的机会很小。如果选择一个大的 B，那么算法将会很慢。采用的实际值取决于当时的情况。

在应用中，我们将采用两个素数乘积的整数，也就是 $n = pq$，但它因数分解难度很大。因此，我们应该确保 $p-1$ 至少有一个大的素数因子，这很容易实现。假设我们希望 p 大约为 100 位，选择一个大的素数 p_0，大约为 10^{40}，考虑一下满足 $kp_0 + 1$ 这种形式的整数，其中 k 约为 10^{60}，和前面一样，用 Miller-Rabin 测试判定 $kp_0 + 1$ 的素数性。一般来说，应该在 100 步之内就能产生预期的值。现在选择一个大的素数 q_0，并按照同样的过程来得到 q，那么 n

$=pq$ 用 $p-1$ 法因数分解的难度很大。

椭圆曲线因数分解法（参考15.3节）给出了 $p-1$ 法的概述，然而，它使用一些接近于 $p-1$ 的随机数并仅仅要求它们中至少一个只有小素数因子。它允许这种方法探测更多的素数 p，而不仅仅是那些满足 $p-1$ 只有小素数因子的素数。

二次筛选法

假设我们想因数分解 $n=3837523$，观察下面的式子：

$$9398^2 \equiv 5^5 \cdot 19 \pmod{3837523}$$
$$19095^2 \equiv 2^2 \cdot 5 \cdot 11 \cdot 13 \cdot 19 \pmod{3837523}$$
$$1964^2 \equiv 3^2 \cdot 13^3 \pmod{3837523}$$
$$17078^2 \equiv 2^6 \cdot 3^2 \cdot 11 \pmod{3837523}。$$

如果我们将上述关系式相乘，得到

$$(9398 \cdot 19095 \cdot 1964 \cdot 17078)^2 \equiv (2^4 \cdot 3^2 \cdot 5^3 \cdot 11 \cdot 13^2 \cdot 19)^2$$
$$2230387^2 \equiv 2586705^2。$$

因为 $2230387 \not\equiv \pm 2586705 \pmod{3837523}$，所以可以通过计算

$$\gcd(2230387-2586705, 3837523)=1093$$

因数分解3837523。另一个因子为3837523/1093 = 3511。

这里是一种查看刚才我们所做计算的方法。首先，生成平方，当这些平方被简化对 $n=3837523$ 取模时，可以写成小素数（在当前的例子中，小于20的素数）乘积的形式，这个素数的集合被称作我们的**因子基（factor base）**。我们将讨论怎样快速地生成这些平方，这些平方的每一个都给出了矩阵中的一行，其中矩阵的元素为素数2,3,5,7,11,13,17,19的指数。例如，关系式 $17078^2 \equiv 2^6 \cdot 3^2 \cdot 11 \pmod{3837523}$ 给出了行：6，2，0，0，1，0，0，0。

除了前述的关系式，假设我们还发现了下面的关系式：

$$8077^2 \equiv 2 \cdot 19 \pmod{3837523}$$
$$3397^2 \equiv 2^5 \cdot 5 \cdot 13^2 \pmod{3837523}$$
$$14262^2 \equiv 5^2 \cdot 7^2 \cdot 13 \pmod{3837523}。$$

可得到矩阵

9398	0	0	5	0	0	0	0	1
19095	2	0	1	0	1	1	0	1
1964	0	2	0	0	0	3	0	0
17078	6	2	0	0	1	0	0	0
8077	1	0	0	0	0	0	0	1
3397	5	0	1	0	0	2	0	0
14262	0	0	2	2	0	1	0	0

现在在这些行中寻找模2的线性相关性。这里是它们中的3个：

1. 1st + 5th + 6th =(6，0，6，0，0，2，0，2) ≡0 (mod 2)
2. 1st + 2nd + 3rd + 4th =(6，4，6，0，2，4，0，2) ≡0 (mod 2)
3. 3rd + 7th =(0，2，2，2，0，4，0，0) ≡0 (mod 2)

当我们有这样一个相关性时，这些数字的乘积产生一个平方。例如，上述3个产生：

1. $(9398 \cdot 8077 \cdot 3397)^2 \equiv 2^6 \cdot 5^6 \cdot 13^2 \cdot 19^2 \equiv (2^3 \cdot 5^3 \cdot 13 \cdot 19)^2$

2. $(9398 \cdot 19095 \cdot 1964 \cdot 17078)^2 \equiv (2^3 \cdot 3^2 \cdot 5^3 \cdot 11 \cdot 13^2 \cdot 19)^2$

3. $(1964 \cdot 14262)^2 \equiv (3 \cdot 5 \cdot 7 \cdot 13^2)^2$

因此，对于不同的 x 和 y 值有 $x^2 \equiv y^2 \pmod n$。如果 $x \not\equiv \pm y \pmod n$，那么 $\gcd(x-y,n)$ 生成 n 的一个非平凡因子。如果 $x \equiv \pm y \pmod n$，那么 $\gcd(x-y,n)=1$ 或 n，所以我们不能得到因数分解。在这 3 个例子中，我们可得到：

1. $3590523^2 \equiv 247000^2$，但 $3590523 \equiv -247000 \pmod{3837523}$

2. $2230387^2 \equiv 2586705^2$ 和 gcd（2230387 − 2586705，3837523）= 1093

3. $1147907^2 \equiv 17745^2$ 和 gcd（1147907 − 17745，3837523）= 1093

现在我们转向基本问题：怎样找到数字 9398，19095，等等？方法是产生比 n 的自乘稍微大一点的平方，所以它们对 n 取模后就比较小，这意味着它们很有可能是小素数的乘积。对于较小的 j 和任意的 i 值，一个简单的方法是考虑形式为 $[\sqrt{in}+j]$ 的数字，这里 $[x]$ 表示小于等于 x 的最大整数。这样一个数的平方近似等于 $in + 2j\sqrt{in} + j^2$，也就是说近似为 $2j\sqrt{in}+j^2$ 对 n 取模的余数，只要 i 不太大，这个数就是很小的，因此很有可能它是小素数的乘积。

例如，在前面的计算中，有 $8077 = [\sqrt{17n}+1]$ 和 $9398 = [\sqrt{23n}+4]$。

刚才用的方法是当前众多最好的因数分解法的基础，主要一步是生成同余关系

$$x^2 \equiv \text{小素数的乘积。}$$

一个上述方法的改进版被称作二次筛选法（the quadratic sieve)，它是一种近代的方法，又叫数域筛选法，它是用更加复杂的技术来生成这样的关系式，在许多情况下它是比较快的。

和以前一样，一旦有了几个同余关系，我们就把它们转换成矩阵，如果在矩阵中行比列多，那么我们应该确保在这些行中有模 2 的线性相关的关系，这导致了同余关系 $x^2 \equiv y^2 \pmod n$。当然，在前面考虑的 $1\text{st} + 5\text{th} + 6\text{th} \equiv 0 \pmod 2$ 的例子中，可以以 $x \equiv \pm y$ 结束，在这个例子中我们没有得到因数分解，但这种情况预计在很多时候会发生。所以如果我们有足够的关系——例如，如果行比列多几个——那么我们应该有一个关系，它可以产生 $x^2 \equiv y^2$，其中 $x \not\equiv \pm y$。在这个例子中，$\gcd(x-y,n)$ 是 n 的一个非平凡因子。

在 20 世纪末期，在因数分解上有了巨大的进步。这部分归功于计算机的发展，部分归功于算法的改进。主要的推动力是因数分解在密码学中的应用，特别是 RSA 算法。表 6.1 给出了各年的因数分解记录（根据十进制数字的位数）。

表 6.1　因数分解记录

年	数字位数
1964	20
1974	45
1984	71
1994	129
1999	155

6.5 RSA 挑战

当 1977 年 RSA 算法首次被公布时，作者提出了下面的挑战。

令 RSA 模数为

$n=$

$11438162575788886766923577997614661201021829672124236256256184293570693524573389783059712356395870505898907514759929002687954354 1$

令加密指数 $e=9007$，密文为

$c=$

$96869613754622061477140922254355882905759991124574319874695120930816298225145708356931476622883989628013391990551829945157815154$。

求解原信息。

得到明文的惟一已知方法是因数分解 n。在 1977 年，预计当时的因数分解法将花费 4×10^{16} 年才能完成，所以作者觉得很安全，并许诺不管谁在 1982 年 4 月 1 日前破解了这个信息，将获得 100 美元。然而，技术已经进步了，1994 年，Atkins，Graff，Lenstra 和 Leyland 成功地因数分解了 n。

他们用了 524339 个“小”素数，也就是那些小于 16333610 的素数，加上他们允许因数分解包括 16333610 和 2^{30} 之间的两个“大”素数。允许大素数的思想如下所示：如果一个大素数 q 出现在两个不同的关系式中，那么它们能够相乘而生成一个关于 q 的平方的关系式，用 $q^{-2} \pmod n$ 乘产生一个仅仅关于小素数的关系式。用同样的方法，如果有几个关系式，每一个都有相同的两个大素数，那么类似的过程可生成一个仅仅关于小素数的关系式。生日攻击（参考 8.4 节）意味着应该存在几例一个大素数出现在多个关系式中的情况。

600 个人总共用了 1600 台计算机在业余时间工作，找到了预期类型的同余关系，它们通过电子邮件传送给中心机，中心机去除重复的，并把结果存在一个大矩阵中。7 个月后，他们得到了一个有 524339 列和 569466 行的矩阵。幸运的是，这个矩阵是稀疏矩阵，也就是说矩阵的大多数元素都是 0，所以它能够被有效地存储。利用高斯消元法将这个矩阵简化成有 188160 列和 188614 行的非稀疏矩阵，这花费了不到 12 个小时。他们又用了另外 45 个小时进行计算，找到了 205 个依赖关系，前 3 个产生了 n 的平凡因子，但第 4 个生成了因子

$p=3490529510847650949147849619903898133417764638493387843990820577$，

$q=32769132993266709549961988190834461413177642967992942539798288533$。

计算 $9007^{-1} \pmod{(p-1)(q-1)}$ 得到解密指数

$d=1066986143685780244428687713289201547807099066339378628012262244966310631259117744708733401685974623065539685445132771090536060 95$。

计算 $c^d \pmod n$ 生成明文信息

200805001301070903002315180419000118050019172105011309190800151919090618010705。

当用 $a = 01, b = 02, \ldots$，空格 $= 00$ 把它转变回字母后，信息为

the magic words are squeamish ossifrage

（神经质的鱼鹰是一种过于敏感的鹰；正是因为选择了这条信息，所以没有人能够通过猜测明文和观察加密后的密文来解密它）。要得到更多关于这个因数分解的信息，请参考［Atkins et al］。

6.6 协议验证上的应用

国家 A 和 B 已经签署了禁止核试验协议，现在每一方都想确认另一方没有试验任何炸弹。例如，国家 A 怎样用地震数据来监控国家 B 呢？国家 A 想把传感器放到国家 B，它能把数据传回国家 A。出现了两个问题。

1. 国家 A 想确认国家 B 没有修改数据。

2. 国家 B 想在信息被传回前查看它，以确认没有其他的信息（如间谍数据）正在被传输。

这些表面上相矛盾的要求可以通过反向 RSA 解决。首先，A 选择 $n = pq$ 为两个大素数的乘积，选择加密和解密指数 e 和 d，数字 n 和 e 公开给 B，但 p，q 和 d 保密，传感器（它被埋在地下很深的地方，假设是防篡改的）收集数据 x，并用 d 把 x 加密为 $y \equiv x^d \pmod n$。x 和 y 都首先被传送到国家 B，由国家 B 核查 $y^e \equiv x \pmod n$。如果成立，那么它就知道加密信息 y 与数据 x 是相应的，并继续把组合 x，y 传送给国家 A；然后国家 A 也核查 $y^e \equiv x \pmod n$，如果成立，那么 A 就能确定数字 x 没有被修改，因为一旦 x 被选定，那么为了求 y 而解 $y^e \equiv x \pmod n$ 与破解 RSA 信息 x 是一样的，而且要确信这样做难度是很大的。当然，B 首先能够选择一个数 y，然后令 $x \equiv y^e \pmod n$，但是 x 可能不是一条有意义的信息，所以 A 就会意识到某些信息被改变了。

前面的方法本质上是 RSA 的数字签名方案，它将在 8.1 节中进行讨论。

6.7 公钥概念

1976 年，Diffe 和 Hellman 描述了公钥密码体制的概念，虽然那时这个概念并没有公开（正如在本章的介绍中所提到的那样，英国密码署 CESG 的 Clifford Cocks 于 1973 年就已经发明了 RSA 的秘密版）。在本节中，我们给出了公钥体制的基本理论。

除了 RSA，还有另外几个公钥密码体制的实现，在后面的章节中我们将描述其中的两个。一个应归功于 ElGamal，它基于解决离散对数的困难性；另一个应归功于 McEliece，它使用的是纠错码。也有基于渐缩问题的公钥系统，本书中不涉及它们；一些版本已经被破解了，一般认为它们比 RSA 和 ElGamal 这样的体制要弱。

公钥密码体制（public key cryptosystem）是由几个部分组成的。首先，包括可能信息的集合 M（可能的明文和密文），还包括密钥的集合 K。这些并不是确定的加密/解密密钥；在 RSA 中，密钥 k 是一个满足 $ed \equiv 1 \pmod{\phi(n)}$ 的三元组（e，d，n）。对于每一个密钥 k，

有一个加密函数 E_k 和解密函数 D_k，通常，E_k 和 D_k 被假定从 M 映射到 M，虽然可能有允许明文和密文来自不同集合的各种变化，这些部分必须满足下列要求。

1. 对于每一个 $m \in M$ 和每一个 $k \in K$，$E_k(D_k(m)) = m$ 和 $D_k(E_k(m)) = m$。

2. 对于每一个 m 和 k，$E_k(m)$ 和 $D_k(m)$ 的值很容易计算出来。

3. 对于几乎每一个 $k \in K$，如果某人仅仅知道函数 E_k，那么找到一个算法计算出 D_k 在计算上是不可行的。

4. 已知 $k \in K$，很容易找到函数 E_k 和 D_k。

要求（1）说明加密和解密互相抵消了；要求（2）是必需的，否则有效地加密和解密是不可能的。由于（4），一个用户能够从 K 中选择一个秘密的随机数 k 并得到函数 E_k 和 D_k，要求（3）使该体制成为公开密钥。因为很难确定 E_k 和 D_k，所以公开 E_k 而不危及系统的安全性是可能的。

让我们看看 RSA 是怎样满足这些要求的。信息空间能够转换成所有的非零整数，正如我们前面所提到的，RSA 的密钥是一个三元组 $k = (e, d, n)$，加密函数为

$$E_k(m) = m^e \pmod n,$$

其中如果 $m \geq n$，那么我们就把 m 分成组。解密函数为

$$D_k(m) = m^d \pmod n,$$

如果有必要再一次把 m 分成组。知道了 k，函数 E_k 和 D_k 马上就可以确定了（要求（4）），并且很容易计算（要求（2））。由于 $ed \equiv 1 \pmod{\phi(n)}$，它们是互逆的，所以满足要求（1）。如果知道了 E_k，这意味着知道了 e 和 n，那么我们看到确定 d（大概）在计算上是不可行的，因此确定 D_k 也不可能。因此，要求（3）（大概）也是满足的。

一旦一个公钥体制被建立，每一个用户都产生了一个密钥 k 并确定了 E_k 和 D_k，加密函数 E_k 被公开，而 D_k 保密，如果发生了冒名顶替的问题，那么真实的授权能够用来分配和验证密钥。

在对称体制中，鲍勃能够确信一条被成功解密的信息一定来自于艾丽斯（确定是一组授权用户）或某个拥有艾丽斯密钥的人，仅仅给了艾丽斯这个密钥，所以没有别人能够生成这个密文。然而，由于鲍勃自己可能已经简单地加密了这条信息，所以艾丽斯能够否认传送了这条信息。因此，鉴别是容易的（如果鲍勃自己没有忘记这条信息，那么他知道信息来自艾丽斯），但反拒绝不容易（参考 1.2 节）。

在公钥系统中，任何人能够加密一条信息并传送给鲍勃，所以他并不知道信息来自哪里，他的确不能证明信息来自艾丽斯。因此，对于鉴别和反拒绝需要更多的措施。虽然这样，通过下面讲述的内容，可以知道这些目标是很容易实现的。

艾丽斯以她的信息 m 开始并计算 $E_{k_b}(D_{k_a}(m))$，其中 k_a 是艾丽斯的密钥，k_b 是鲍勃的密钥。那么鲍勃能够用 D_{k_b} 解密而得到 $E_{k_a}(m)$，他用公共可用的 E_{k_a} 来得到 $E_{k_a}(D_{k_a}(m)) = m$，由于没有别人能够计算 $D_{k_a}(m)$，所以鲍勃知道信息一定来自于艾丽斯。由于同样的原因，艾丽斯不能否认传送了这条信息。当然，所有这些都假定大多数随机信息是没有意义的，所以一个随机的符号串解密成一条有意义的信息是不可能的，除非这个串是某些有意义信息的加密。

这些鉴别方法的具体版本将在关于数字签名的第 8 章中进行讨论。

用满足某些属性的单向函数来构成一个公钥密码体制是可能的。令 $f(m)$ 表示一个可逆

的单向函数，这意味着 $f(x)$ 是很容易计算的，但如果已知 y，那么找到满足 $y=f(x)$ 的惟一 x 值在计算上是不可行的。现在假设 $f(x)$ 有一个陷门（**trapdoor**），这表示对于 x 有一个简单的方法可以求出 $y=f(x)$，但只有函数的设计者才知道额外的信息，而且，函数设计者以外的人来确定这个陷门信息在计算上是不可行的。如果有一个带陷门的非常大的单向函数族，那么它们能够用来形成一个公钥密码体制。每一个用户用只有自己知道陷门的一种方法才能从这个族中生成函数，然后用户的函数作为公开的加密算法被公布。当艾丽斯想传送信息 m 给鲍勃时，她使用函数 $f_b(x)$，并计算 $y=f_b(m)$，艾丽斯传送 y 给鲍勃，因为鲍勃知道 $f_b(x)$ 的陷门，所以他能解 $y=f_b(m)$，从而得到 m。

在 RSA 中，对于适当的 n 和 e，函数 $f(x)=x^e \pmod n$ 形成了单向函数族，陷门信息是 n 的因数分解。在 ElGamal 体制（7.4 节）中，单向函数从求对一个素数取模的幂得到，陷门信息是一个已知的离散对数。在 McEliece 体制（16.10 节）中，陷门信息是为某个线性二进制码找到最接近的码字（"纠错"）的一种有效方法。

6.8 习　题

1. 密文 5859 是用 $n=11413$ 和 $e=7467$ 从 RSA 算法中得到的。使用因数分解 $11413=101\cdot 113$ 找到明文。

2. 令 p 表示一个大素数。假设对于某个加密指数 e，你通过计算 $y\equiv x^e \pmod p$ 来加密信息 x。试问怎样找到一个满足 $y^d\equiv x \pmod p$ 的解密指数 d 呢？

3. 令 p 表示一个大素数。艾丽斯想传送信息 m 给鲍勃，其中 $1\le m\le p-1$，艾丽斯和鲍勃选择整数 a 和与 $p-1$ 相关的素数 b，艾丽斯计算 $c\equiv m^a \pmod p$ 并传送 c 给鲍勃，鲍勃计算 $d\equiv c^b \pmod p$ 并把 d 传回给艾丽斯。因为艾丽斯知道 a，所以她得到满足 $aa_1\equiv 1 \pmod{p-1}$ 的 a_1，然后她计算 $e\equiv d^{a_1} \pmod p$ 并传送 e 给鲍勃。解释一下为了得到 m，鲍勃现在必须做什么，并证明这种做法有效。

4. Naive Nelson 用 RSA 收到了一个单独的密文 c，它与信息 m 相对应，公开模数为 n，公开加密指数为 e。因为他仅仅使用过一次这个体制，感觉心虚，所以他同意只要某人传给他的密文不是 c，他就对它加密并将结果传回给那个人。险恶的伊芙给他传送密文 $2^e c \pmod n$。证明这个密文怎样使伊芙得到 m。

5. 为了增强安全性，鲍勃选择 n 和两个加密指数 e_1，e_2。他请求艾丽斯通过首先计算 $c_1\equiv m^{e_1} \pmod n$，然后加密 c_1 得到 $c_2\equiv c_1^{e_2} \pmod n$ 的方法来加密她的信息 m 传给他，随后艾丽斯传送 c_2 给鲍勃。这个双指数加密比单指数加密增强安全性了吗？请说明原因。

6. 令 p 和 q 表示不同的奇素数，并且 $n=pq$。假设整数 x 满足 $\gcd(x,pq)=1$。

(a) 证明 $x^{\frac{1}{2}\phi(n)}\equiv 1 \pmod p$ 和 $x^{\frac{1}{2}\phi(n)}\equiv 1 \pmod q$。

(b) 用 (a) 证明 $x^{\frac{1}{2}\phi(n)}\equiv 1 \pmod n$。

(c) 用 (b) 证明如果 $ed\equiv 1 \pmod{\frac{1}{2}\phi(n)}$，那么 $x^{ed}\equiv x \pmod n$。（这证明了在 RSA 中我们能够用 $\frac{1}{2}\phi(n)$ 代替 $\phi(n)$ 工作。）

7. 指数 $e=1$ 和 $e=2$ 不应该用在 RSA 中，为什么？

8. 假设在一个网络上有两个用户，令他们的 RSA 模数为 n_1 和 n_2，且 n_1 不等于 n_2。如果已知 n_1 和 n_2 是不相关的素数，你怎样破解他们的系统？

9. 你正在尝试因数分解 $n=642401$，假设你发现

$$516107^2 \equiv 7 \pmod{n}$$

和

$$187722^2 \equiv 2^2 \cdot 7 \pmod{n}。$$

用这个信息因数分解 n。

10. 假设两个用户艾丽斯和鲍勃有相同的 RSA 模数 n，并假设他们的加密指数 e_A 和 e_B 是相关的素数。Charles 想传送信息 m 给艾丽斯和鲍勃，所以他加密 m 得到 $c_A\equiv m^{e_A}$ 和 $c_B\equiv m^{e_B} \pmod{n}$。说明如果伊芙截取了 c_A 和 c_B，她如何得到 m？

11. 假设艾丽斯像下面这样运用 RSA 方法。她以由几个字母组成的信息开始，并赋值 $a=1,b=2,\ldots,z=26$，然后她单独地加密每个字母。例如，如果信息为 *cat*，她计算 $3^e \pmod{n}, 1^e \pmod{n}$ 和 $20^e \pmod{n}$，然后再传送加密信息给鲍勃。解释不用因数分解 n，伊芙怎样才能获得信息。特别地，假设 $n=8881$，$e=13$，伊芙截取的信息如下：

4461　794　2015　2015　3603。

不用因数分解 8881，请找出信息内容。

12. 证明：如果 $x^2\equiv y^2 \pmod{n}$ 且 $x\not\equiv \pm y \pmod{n}$，那么 $\gcd(x+y,n)$ 是 n 的一个非平凡因子。

13. 令 $n=pq$ 表示两个不同素数的乘积。

(a) 令 m 表示 $\phi(n)$ 的倍数。证明：如果 $\gcd(a,n)=1$，那么 $a^m\equiv 1 \pmod{p}$ 和 $\pmod{q}$。

(b) 假设 m 如（a）中所述，令 a 是任意值（可能 $\gcd(a,n)\neq 1$）。证明 $a^{m+1}\equiv a \pmod{p}$ 和 $\pmod{q}$。

(c) 令 e 和 d 是模数为 n 的 RSA 的加密和解密指数。证明对于所有的 a，$a^{ed}\equiv a \pmod{n}$。这表明我们不必为了用 RSA 而假定 $\gcd(a,n)=1$。

(d) 如果 p 和 q 比较大，为什么对于随机选择的 a，$\gcd(a,n)$ 可能为 1？

14. 假设 $n=pqr$ 是 3 个不同素数的乘积，一个 RSA 型的方案在这种情况下有效吗？特别地，e 和 d 将满足什么关系？

注意：用 3 个素数代替两个素数似乎没有任何优点。一些因数分解法的执行时间取决于最小素数因子的大小，因此，如果使用 3 个素数，为了与用两个素数获得相同的安全级，n 的大小一定增加了。

15. 令 $p=7919, q=17389, e=66909025$。计算证明 $e^2\equiv 1 \pmod{(p-1)(q-1)}$。艾丽斯决定用模数为 $n=pq$、指数为 e 的 RSA 加密信息 $m=12345$。因为她想使加密更安全，所以她再一次用 n 和 e（这样她对原始明文进行了两次加密）加密了密文。她传送的最终密文是什么？不使用计算机，证明你的答案是正确的。

16. 假设你正在使用 RSA（模数 $n=pq$，加密指数为 e），但是你决定限定你的信息为满足 $m^{1000}\equiv 1 \pmod{n}$ 的数字 m。

(a) 证明：如果 d 满足 $de\equiv 1 \pmod{1000}$，那么 d 作为一个解密指数对这些信息是有效的。

(b) 假定 p 和 q 都同余于 1 模 1000。确定有多少信息满足 $m^{1000} \equiv 1 \pmod n$。当 r 是一个同余于 1 模 1000 的素数时，你可以假定并使用 $m^{1000} \equiv 1 \pmod r$ 有 1000 个解这个论据。

17. 假设鲍勃的加密公司生产了两台机器 A 和 B，它们都应该是用同样模数 $n = pq$ 的 RSA 算法实现的，其中 p 和 q 是未知的，两台机器也都用了相同的加密指数 e。每台机器收到一个信息 m 并输出一个应该为 $m^e \pmod n$ 的密文，机器 A 总是生成正确的输出，然而，由于执行和硬件错误，机器 B 总是输出满足 $c \equiv m^e \pmod p$ 和 $c \equiv m^e + 1 \pmod q$ 的密文 $c \pmod n$。你怎样用机器 A 和 B 找到 p 和 q？（参考上机题 11 关于这样的情况如何发生的讨论。）

6.9 上 机 题

注意：下列问题中的许多数字对于没有 Maple Kernel 帮助的 MATLAB 来说太大了。

1. Paul Revere 的一个在麻省理工学院广播塔中的朋友说，如果（英国人）由陆路到来他就传送 *one*，如果由海路到来他就传送 *two*。因为他们知道在波士顿地区 RSA 将被发明，所以他们决定信息应该用 $n = 712446816787$ 和 $e = 6551$ 的 RSA 加密。Paul Revere 收到的密文是 273095689186，请问明文是什么？

2. 在一个 RSA 密码体制中，假设已经知道 $n = 718548065973745507$，$e = 3449$，$d = 543546506135745129$，请因数分解 n。

3. 选择两个 30 位的素数 p 和 q，以及一个加密指数 e，加密明文 *cat*, *bat*, *hat*, *encyclopedia*, *antidisestablishmentarianism* 中的每一个。只看密文你能够辨别出前 3 个明文只有一个字母不同吗？或者后两个明文比前 3 个要长得多？

4. 用 $p-1$ 法因数分解 618240007109027021。

5. 用 $p-1$ 法因数分解 8834884587090814646372459890377418962766907（数字存储为 n1）。

6. 令 $n = 537069139875071$，假设已知

$$859753244431662^2 \equiv 462436106261^2 \pmod n,$$

试因数分解 n。

7. 令 $n = 84047 \cdot 65497$，找到满足 $x^2 \equiv y^2 \pmod n$ 但 $x \not\equiv \pm y \pmod n$ 的 x 和 y。

8. (a) 假设已知

$$33335^2 \equiv 670705093^2 \pmod{670726081},$$

利用这条信息来因数分解 670726081。

(b) 假设已知 $3^2 \equiv 670726078^2 \pmod{670726081}$，这条信息为什么不能帮你因数分解 670726081？

9. 假设已知

$$2^{958230} \equiv 1488665 \pmod{3837523},$$
$$2^{1916460} \equiv 1 \pmod{3837523}。$$

怎样利用这条信息来因数分解 3837523？注意指数 1916460 是指数 958230 的两倍。

10. (a) 假设用在 RSA 算法中的素数 p 和 q 是连续的素数。怎样因数分解 $n = pq$？

(b) 密文 10787770728 是用 $n = 10993522499$ 和 $e = 113$ 加密的，选择 n 的因子 p 和 q 以

满足 $q - p = 2$，试解密这个信息。

（c）下面的密文 c 是用指数 e 对 n 取模加密的：

$$n = 152415787501905985701881832150835089037858868621211004433$$

$$e = 9007$$

$$c = 141077461765569500241199505617854673388398574333341423525。$$

n 的素数因子 p 和 q 是连续的素数。试解密这个信息。（n 存储为 *naive*，c 存储为 *cnaive*。）

11. 令 $p = 123456791$，$q = 987654323$，$e = 127$，假设信息为 $m = 14152019010605$。

（a）计算 $m^e \pmod p$ 和 $m^e \pmod q$；那么用中国剩余定理来结合它们得到 $c \equiv m^e \pmod{pq}$。

（b）改变 $m^e \pmod p$ 的一位（例如，这可能是由某些辐射引起的）。现在把它与 $m^e \pmod q$ 结合起来，得到一个 $m^e \pmod{pq}$ 的不正确的值 f，计算 $\gcd(c - f, pq)$。为什么可以因数分解 pq？（a）中计算 $m^e \pmod{pq}$ 的方法很有吸引力，因为它不像直接对 pq 取模那样要求大的多倍精度运算。然而，正如问题（b）显示的，如果一个攻击者能够引起任意一位出错，那么 pq 就能够因数分解。

12. 假设 $p = 76543692179$，$q = 343434343453$，$e = 457$，密文 $c \equiv m^e \pmod{pq}$ 被传输，但传送过程中产生了一个错误。收到的密文为 2304329328016936947195。接收者能够确定收到的位数是正确的，但最后一位丢失了，试确定丢失的位并解密信息。

13. 用 $a = 2$ 的 Miller-Rabin 测试判断 38200901201 的素数性，然后用 $a = 3$ 判断。注意第一次判断认为 38200901201 可能是素数，而第二次判断认为它是合数。像 38200901201 这样通过数 a 的 Miller-Rabin 测试的合数称作**强 a 伪素数（strong a - pseudoprime）**。

14.（a）假设有 3 个用户，其素模数 n_1，n_2，n_3 两两相关，但是假设他们的加密指数都为 $e = 3$，如果相同的信息被传送给他们中的每一个并且你截获了密文，你怎样确定这个信息（不用因数分解）？

（b）假设

$$n_1 = 2469247531693, n_2 = 11111502225583, n_3 = 44444222221411,$$

相应的密文是

$$359335245251, 10436363975495, 5135984059593。$$

这些都是用 $e = 3$ 加密的，试获取这个信息。（提示：中国剩余定理会有所帮助，所以会用到 $x^{1/3}$。）

第7章 离散对数

7.1 离散对数

在 RSA 算法中，我们了解了基于因数分解的加密体制。另外还有一个数论问题，称为离散对数，它有类似的应用。

选择一个素数 p，设 α 和 β 为非 0 的模 p 整数，令

$$\beta \equiv \alpha^x \pmod p,$$

求 x 的问题称为**离散对数问题**（**discrete logarithm problem**）。如果 n 是满足 $\alpha^n \equiv 1 \pmod p$ 的最小正整数，假设 $0 \leqslant x < n$，我们记

$$x = L_\alpha(\beta)$$

并称之为与 α 相关的 β 的离散对数（素数 p 可从符号中忽略）。

举一个例子，设 $p = 11, \alpha = 2$，因为 $2^6 \equiv 9 \pmod{11}$，有 $L_2(9) = 6$。当然，$2^6 \equiv 2^{16} \equiv 2^{26} \equiv 9 \pmod{11}$，所以我们认为能够取 6，16，26 中的任何一个作为离散对数，但所要的是最小的非负值，那么只能是 6。注意，在本例中我们能够将离散对数定义为 6 mod 10 的同余类。在某些情况下，这样的定义会更自然，但存在一些应用，很容易找到一个数而并不是同余类。

通常，α 被当作模 p 本原根，这表明每一个 β 都是 $\alpha \pmod p$ 的一个幂。如果 α 不是本原根，那么就不会为 β 值定义离散对数。

给定一个素数 p，在很多情况下很容易找到它的本原根，见练习 3.10。

离散对数在很多情况下和一般对数一样，特殊情况下，如果 α 是模 p 本原根，那么

$$L_\alpha(\beta_1\beta_2) \equiv L_\alpha(\beta_1) + L_\alpha(\beta_2) \pmod{p-1}$$

（见练习 3）。

当 p 很小的时候，通过穷尽搜索法很容易计算离散对数。然而，当 p 较大时，这种方法就行不通了。随后我们会给出一些攻击离散对数问题的方法。但是，一般情况下都认为离散对数计算起来很难，而这也正是后面几种加密体制的基础。

离散对数能够被计算的最大素数长度与能够被因数分解的最大整数长度几乎是相等的（它们两个所涉及到的计算对这些长度的任意数都是有效的；为满足某些特殊应用需要专门挑选整数，这样离散对数的计算和因数分解的工作量对某些特别的数就非常大）。2001 年，离散对数能够计算 110 位的素数，这是当时离散对数能够计算的记录。后来因数分解的记录上升到 155 位。

如果 $f(x)$ 很容易计算，那么函数 $f(x)$ 就称为**单向函数(one-way function)**，但是，已知 y，就很难由 $f(x)=y$ 求出 x。模的幂运算是这种函数的一个例子，计算 $\alpha^x \pmod p$ 很容易，但解出 $\alpha^x \equiv \beta$ 中的 x 通常是很困难的；大素数的乘法也被认为是（可能的）单向函数：很容易计算素数乘法，但很难根据最后的乘积得到相乘之前的素数。总之，单向函数有许多密码学方面的应用。

7.2 离散对数的计算

在这一节中，我们将介绍几种计算离散对数的方法。另外一种有用的方法，即生日攻击方法，将在 8.4 节中进行讨论。

简单地说，设 α 是模 p 本原根，那么 $p-1$ 就是使 $\alpha^n \equiv 1 \pmod p$ 成立的最小正指数 n，这隐含着：

$$\alpha^{m_1} \equiv \alpha^{m_2} \pmod p \Leftrightarrow m_1 \equiv m_2 \pmod{p-1}。$$

假设

$$\beta \equiv \alpha^x, 0 \leqslant x < p-1,$$

我们要求 x。

首先，很容易求得 $x \pmod 2$。注意：

$$(\alpha^{(p-1)/2})^2 \equiv \alpha^{p-1} \equiv 1 \pmod p,$$

所以 $\alpha^{(p-1)/2} \equiv \pm 1 \pmod p$（见练习 3.4）。然而，$p-1$ 被认为是产生 $+1$ 的最小指数，所以必须有：

$$\alpha^{(p-1)/2} \equiv -1 \pmod p。$$

因为从 $\beta \equiv \alpha^x \pmod p$ 开始，两边同时升幂到 $(p-1)/2$，得到：

$$\beta^{(p-1)/2} \equiv \alpha^{x(p-1)/2} \equiv (-1)^x \pmod p$$

因此，如果 $\beta^{(p-1)/2} \equiv +1$，那么 x 是偶数；否则，x 是奇数。

例如：假设我们要计算 $2^x \equiv 9 \pmod{11}$，因为

$$\beta^{(p-1)/2} \equiv 9^5 \equiv 1 \pmod{11},$$

所以 x 必须是偶数。事实上，前面我们已经知道了 $x=6$。 ■

7.2.1 Pohlig-Hellman 算法

Pohlig 和 Hellman 将上述算法扩展，得到了当 $p-1$ 只有小的素数因子时计算离散对数的方法。假设

$$p-1 = \prod_i q_i^{r_i}$$

是 $p-1$ 的因数分解，设 q^r 是其中的一个因子，我们将计算 $L_\alpha(\beta) \pmod{q^r}$。如果每一个 $q_i^{r_i}$ 都能解出，那么根据中国剩余定理，就不难求得离散对数。

记：

$$x = x_0 + x_1 q + x_2 q^2 + \cdots, 其中 0 \leqslant x_i \leqslant q-1。$$

我们可计算出系数 x_0，x_1，$\ldots x_{r-1}$，然后就可以得到 $x \bmod q^r$。注意，

$$x\left(\frac{p-1}{q}\right) = x_0\left(\frac{p-1}{q}\right) + (p-1)(x_1 + x_2 q + x_3 q^2 + \cdots)$$

$$= x_0\left(\frac{p-1}{q}\right) + (p-1)n,$$

其中，n 是一个整数。从 $\beta \equiv \alpha^x$ 开始，提升两边到幂 $(p-1)/q$，得到

$$\beta^{(p-1)/q} \equiv \alpha^{x(p-1)/q} \equiv \alpha^{x_0(p-1)/q}(\alpha^{p-1})^n \equiv \alpha^{x_0(p-1)/q} \pmod p。$$

最后一个同余式是费尔马定理的序列：$\alpha^{p-1} \equiv 1 \pmod p$。为了求 x_0，只需进行下列幂运算：

$$\alpha^{k(p-1)/q} \pmod p, k = 0, 1, 2, \ldots, q-1,$$

直到它们中有一个能够得到 $\beta^{(p-1)/q}$。那么，$x_0 = k$。注意到，由于 $\alpha^{m_1} \equiv \alpha^{m_2} \Leftrightarrow m_1 \equiv m_2 \pmod{p-1}$，并且因为指数 $k(p-1)/q$ 对 $p-1$ 取模是不同的，所以只有一个 k 可以得到这个结果。

用这个思想继续计算，就可以得到余下的系数。假设有 $q^2 \mid p-1$，设

$$\beta_1 \equiv \beta\alpha^{-x_0} \equiv \alpha^{q(x_1 + x_2 q + \cdots)} \pmod p,$$

提升两边的幂到 $(p-1)/q^2$，得到：

$$\beta_1^{(p-1)/q^2} \equiv \alpha^{(p-1)(x_1 + x_2 q + \cdots)/q}$$

$$\equiv \alpha^{x_1(p-1)/q}(\alpha^{p-1})^{x_2 + x_3 q + \cdots}$$

$$\equiv \alpha^{x_1(p-1)/q} \pmod p,$$

最后一个同余式是由费尔马定理得出的。因为分数指数的运算会带来一些问题，所以我们不能够计算 $\beta_1^{(p-1)/q^2}$ 为 $(\beta_1^{p-1})^{1/q^2}$。注意我们所用到的指数都是整数。

为了求 x_1，只需要进行下面的幂运算：

$$a^{k(p-1)/q} \pmod p, \qquad k = 0, 1, 2, \ldots, q-1,$$

直到其中的一个 k 代入之后能够得到 $\beta_1^{(p-1)/q^2}$，那么 $x_1 = k$。

如果 $q^3 \mid p-1$，设 $\beta_2 = \beta_1\alpha^{-x_1 q}$，提升两边到 $(p-1)/q^3$ 次方，可以求出 x_2，用这种方法继续计算，直到找到 q^{r+1} 不能整除 $p-1$。由于分数的幂运算是没有意义的，所以我们就不必再计算下去。但是我们已经求得 x_0，x_1，$\ldots$，x_{r-1}，所以我们也能够得到 $x \bmod q^r$。

对所有 $p-1$ 的素数因子，重复上述过程，可得到对所有 i，$x \bmod q_i^{r_i}$ 的值。中国剩余定理允许我们将这些组合成 $x \bmod p-1$ 的同余式。因为 x 的范围是 $0 \leqslant x < p-1$，这就可以确定 x 的值。

例如：令 $p = 41$，$\alpha = 7$，$\beta = 12$，求解：

$$7^x \equiv 12 \pmod{41}。$$

注意到

$$41 - 1 = 2^3 \cdot 5,$$

首先，设 $q = 2$，我们来求 $x \bmod 2^3$。记 $x \equiv x_0 + 2x_1 + 4x_2 \pmod 8$，开始

$$\beta^{(p-1)/2} \equiv 12^{20} \equiv 40 \equiv -1 \pmod{41},$$

且

$$\alpha^{(p-1)/2} \equiv 7^{20} \equiv -1 \pmod{41}。$$

因为

$$\beta^{(p-1)/2} \equiv (\alpha^{(p-1)/2})^{x_0} \pmod{41},$$

我们可以得到 $x_0 = 1$。下一步，

$$\beta_1 \equiv \beta\alpha^{-x_0} \equiv 12 \cdot 7^{-1} \equiv 31(\bmod 41),$$

又有

$$\beta_1^{(p-1)/2^2} \equiv 31^{10} \equiv 1(\bmod 41)。$$

因为

$$\beta_1^{(p-1)/2^2} \equiv (\alpha^{(p-1)/2})^{x_1}(\bmod 41),$$

我们可以得到 $x_1 = 0$。继续计算，可以得到：

$$\beta_2 \equiv \beta_1\alpha^{-2x_1} \equiv 31 \cdot 7^0 \equiv 31(\bmod 41),$$

和

$$\beta_2^{(p-1)/q^3} \equiv 31^5 \equiv -1 \equiv (\alpha^{(p-1)/2})^{x_2}(\bmod 41),$$

所以 $x_2 = 1$。，可以得到：

$$x \equiv x_0 + 2x_1 + 4x_2 \equiv 1 + 4 \equiv 5(\bmod 8)。$$

现在，设 $q = 5$，求 $x \bmod 5$。已知：

$$\beta^{(p-1)/5} \equiv 12^8 \equiv 18(\bmod 41)$$

和

$$\alpha^{(p-1)/q} \equiv 7^8 \equiv 37(\bmod 41)。$$

尝试 k 所有可能的值得到：

$$37^0 \equiv 1,\ 37^1 \equiv 37,\ 37^2 \equiv 16,\ 37^3 \equiv 18,\ 37^4 \equiv 10\ (\bmod 41)$$

所以，37^3是原方程的解，这样，$x \equiv 3(\bmod 5)$。

因为 $x \equiv 5(\bmod 8)$，$x \equiv 3(\bmod 5)$，我们将二者合并可以得到： $x \equiv 13(\bmod 40)$，所以 $x = 13$。这也可以通过计算 $7^{13} \equiv 12(\bmod 41)$ 快速检验。 ■

只要上述算法中的 q 足够小，就能够很快计算出结果。然而，当 q 较大时，计算 $\alpha^{k(p-1)/q}$，$k = 0, 1, 2, ..., q-1$ 不可行，所以这种算法就不再实用。这意味着如果我们想要一个离散对数的计算难度增大，就必须确保 $p-1$ 有一个大的素数因子。

我们还注意到，尽管 $p-1 = tq$ 有一个较大的素数因子 q，但如果 t 是由小素数因子组成的，那么该算法仍然能够求出模 t 的离散对数。为此，经常选取 β 为 α^t，那么离散对数自然就是 $0 \bmod t$，所以离散对数隐藏了模 q 的信息，这是该算法没有预料到的。如果离散对数 x 代表一个未知数（或者说，t 乘以一个未知数），这就意味着一个攻击者不能通过计算 $x \bmod t$ 而获得任何信息，因为没有信息用这种方法隐藏。数字签名算法就用到了这种算法思想，我们在第 8 章会详细讨论。

7.2.2 指数微积分

这种思想类似于因数分解的二次过滤。我们需要再次计算 $\beta \equiv \alpha^x(\bmod p)$，其中，$p$ 是素数，α 是本原根。

首先，有一个预备计算。设 B 是上限值，p_1，p_2，...，p_m是小于 B 的素数，这个素数的集合称为因子基(**factor base**)。对不同的 k 值，计算 $\alpha^k(\bmod p)$，对每一个数字，试着表示为小于 B 的素数的乘积，如果不是这样，就将 α^k舍弃。但是，如果 $\alpha^k \equiv \prod p_i^{a_i}(\bmod p)$，那么

$$k \equiv \sum \alpha_i L_\alpha(p_i)(\bmod p-1)。$$

当我们得到足够这样的关系后，就能对每个 i 求出 $L_\alpha(p_i)$ 。

现在，对任意的整数 r，计算 $\beta\alpha^r \pmod p$。对每一个这样的数，试着将其写成小于 B 的素数的乘积的形式。如果能够写成这种形式，就有 $\beta\alpha^r \equiv \prod p_i^{b_i} \pmod p$ ，即：

$$L_\alpha(\beta) \equiv -r + \sum b_i L_\alpha(p_i) \pmod{p-1}。$$

如果 p 相当大，这种算法的效率是非常高的。也就是说，如果想要离散对数问题的难度增大，p 至少应该有 200 位，或者更多。

例如：设 $p=131$，$\alpha=2$，令 $B=10$，我们用素数 2，3，5，7 进行计算，计算后得到下式：

$$\begin{aligned}
2^1 &\equiv 2 \pmod{131}\\
2^8 &\equiv 5^3 \pmod{131}\\
2^{12} &\equiv 5\cdot 7 \pmod{131}\\
2^{14} &\equiv 3^2 \pmod{131}\\
2^{34} &\equiv 3\cdot 5^2 \pmod{131}。
\end{aligned}$$

所以，

$$\begin{aligned}
1 &\equiv L_2(2) \pmod{130}\\
8 &\equiv 3L_2(5) \pmod{130}\\
12 &\equiv L_2(5)+L_2(7) \pmod{130}\\
14 &\equiv 2L_2(3) \pmod{130}\\
34 &\equiv L_2(3)+2L_2(5) \pmod{130}。
\end{aligned}$$

由第 2 个同余式可得到 $L_2(5) \equiv 46 \pmod{130}$ ，带入第 3 个同余式得到：$L_2(7) \equiv -34 \equiv 96 \pmod{130}$ 。第 4 个同余式只得到 $L_2(3) \pmod{65}$ 的值，因为 $\gcd(2,130) \neq 1$，这就使 $L_2(3) \pmod{130}$ 有两个可能的解，当然我们可以通过试探法决定用哪一个，或者我们可以使用第 5 个同余式去求得 $L_2(3) \equiv 72 \pmod{130}$，这就完成了所有预备计算的步骤。

假设我们要得到 $L_2(37)$ 。用试探法选择一些随机的指数可以得到：$37\cdot 2^{43} \equiv 3\cdot 5\cdot 7 \pmod{131}$，所以

$$L_2(37) \equiv -43 + L_2(3) + L_2(5) + L_2(7) \equiv 41 \pmod{130}。$$

因此，$L_2(37)=41$ 。 ■

当然，一旦预备计算完成之后，就可以用它来计算与素数 p 相同的其他离散对数。

7.2.3 模 4 离散对数的计算

当 $p \equiv 1 \pmod 4$ 时，用 Pohlig-Hellman 算法计算模 4 离散对数速度相当快。但是，计算 $p \equiv 3 \pmod 4$ 时又如何呢？这时 Pohlig-Hellman 算法就不起作用了，因为它需要两边同时提升到幂 $(p-1)/4$ 次方，而这样有可能使指数为分数。事实上，如果我们有一个算法能够对素数 $p \equiv 3 \pmod 4$ 快速计算模 4 离散对数的话，那么就能够用它来快速计算模 p 离散对数。所以，这样的算法是不可能存在的。

我们不能够期望这样的一个算法，有一个哲学上的原因。一般认为，离散函数可以当作是一个模 $p-1$ 数字，所以我们应该能够在离散对数上获得关于 $p-1$ 的 2 的幂的模数的知识。当 $p\equiv 3 \pmod 4$ 时，说明模 4 离散对数这样的问题是没有任何意义的。之所以这样只因为我们规范化离散对数是介于 0 与 $p-2$ 之间的整数，例如 $2^6\equiv 2^{16}\equiv 9 \pmod{11}$。在这个例子中，我们定义 $L_2(9)$ 等于 6；假如我们也允许它为 16，将得到 $L_2(9)$ 的两个值，即 6 和 16，它们模 4 后并不同余。因此，从这一点来讲，说 $L_2(9)$ 模 4 是没有任何意义的。

我们需要下面的推论，这个推论与计算模素数 $p\equiv 3 \pmod 4$ 的平方根的方法十分相似（见 3.9 节）。

推论：设 $p\equiv 3 \pmod 4$ 是素数，令 $r\geq 2$，y 是整数。假设 α 和 γ 是使 $\gamma\equiv\alpha^{2^r y} \pmod p$ 成立的两个非零数字，那么

$$\gamma^{(p+1)/4}\equiv\alpha^{2^{r-1}y} \pmod p。$$

证明：

$$\gamma^{(p+1)/4}\equiv\alpha^{(p+1)2^{r-2}y}\equiv\alpha^{2^{r-1}y}(\alpha^{p-1})^{2^{r-2}y}\equiv\alpha^{2^{r-1}y} \pmod p,$$

最后一步同余式的根据是费尔马定理。 □

设素数 $p\equiv 3 \pmod 4$，α 是本原根。假设我们有一台机器，输入 β，输出 $L_\alpha(\beta) \bmod 4$。从前面我们已经知道，很容易计算 $L_\alpha(\beta) \bmod 2$，所以这台机器所提供的新信息实际上就是离散对数的第二个比特位。

现在假设 $\alpha^x\equiv\beta \pmod p$，$x=x_0+2x_1+4x_2+\cdots+2^n x_n$ 是 x 的二进制扩展表示，使用 $L_\alpha(\beta) \pmod 4$ 机器，可以求得 x_0 和 x_1。假设当 $r\geq 2$ 时，我们已经求得 x_0，x_1，...，x_{r-1}，令

$$\beta_r\equiv\beta\alpha^{-(x_0+\cdots+2^{r-1}x_{r-1})}\equiv\alpha^{2^r(x_r+2x_{r+1}+\cdots)},$$

用这个定理 $r-1$ 次，我们可以发现：

$$\beta_r^{((p+1)/4)^{r-1}}\equiv\alpha^{2(x_r+2x_{r+1}+\cdots)} \pmod p。$$

应用 $L_\alpha \pmod 4$ 机器到这个等式，可以得到 x_r 的值。继续计算可以得到 $x_0, x_1, ..., x_n$ 的值，这样就确定了 x。

这个算法在经过加工之后可以变得效率更高，具体事例［见 Stinson 175 页］。

总之，如果我们认为，对 $p\equiv 3 \pmod 4$ 而言，计算离散对数十分困难，那么计算其他的模 4 离散对数也一样困难。

7.3 比特约定

艾丽斯声称她有方法预知足球比赛的结果，她想把她的方法卖给鲍勃，鲍勃让她通过预测这周末举行的比赛来证明她的方法的正确性，“我不可能这么做”，艾丽斯说：“如果我告诉了你这周末的比赛结果，你就会下赌注，而且不付钱给我。如果你想让我证明我的方法起作用，为什么不让我通过预测上周的比赛来证明？”很显然，问题就在这儿，我们将显示怎样来解决。

这是初始建立。艾丽斯想要发送一个比特 b 给鲍勃，这个位或者是 0，或者是 1。下面

是两个要求：

1. 如果没有艾丽斯的帮助，鲍勃就不能够确定这一位的值。

2. 艾丽斯一旦将某位值发送，就不能够改变它。

艾丽斯的一个办法是将比特放在盒子里，并用自己的锁锁上，然后将盒子发送给鲍勃。当鲍勃想要知道这一位的值的时候，艾丽斯开锁，鲍勃打开盒子。我们想要一种数学方法，能够使得这一位显示时，鲍勃和艾丽斯可以不在一个房间内。

这里有一个解决办法。艾丽斯和鲍勃就一个大的素数 $p \equiv 3 \pmod 4$ 和一个本原根 α 达成一致，艾丽斯选择一个随机数 $x < p-1$，它的第二位 x_1 是 b，她发送 $\beta \equiv \alpha^x \pmod p$ 给鲍勃，我们假定鲍勃不能计算 p 的离散对数。正如最后一部分所指出，这意味着他不能计算模 4 离散对数，更具体地说，他不能确定 $b = x_1$ 的值。当鲍勃想知道 b 的值，艾丽斯发送给他 x 的全部值，而且通过计算 $x \bmod 4$，他求得 b。艾丽斯不可能发送一个与已经用过的那个数值不同的 x 值，因为鲍勃核查 $\beta \equiv \alpha^x \pmod p$，这个方程式只有惟一的 $x < p-1$ 的解。

返回到足球比赛。对每场比赛，如果预测这个队将赢，就发送 $b = 1$，否则发送 $b = 0$，比赛结束后，艾丽斯将二进制位展示给鲍勃，让鲍勃看她的预测是否正确。用这种方法，鲍勃不能在比赛前收到信息并从中获益，而且一旦比赛开始，艾丽斯就不能改变她预测的结果。

7.4　ElGamal 公钥体制

在第 6 章，我们学习了基于因数分解的公钥体制。同样，我们也可以设计这样一个体制，其安全性依赖于计算离散对数的困难性。这项工作在 1985 年由 ElGamal 完成。这个体制并不是很符合在第 6 章最后给出的公钥的定义，因为参数的设置不相同。然而，这个技术点我们并不关心。

艾丽斯想要发送一个信息 m 给鲍勃，鲍勃选择一个大的素数 p 和一个本原根 α，假设 m 是一个整数，且 $0 \leq m < p$。如果 m 是较大的数，就把它分解成小的数，鲍勃也选择一个秘密的整数 a 并且计算 $\beta \equiv \alpha^a \pmod p$。信息 (p, α, β) 被公开，它就是鲍勃的公钥。艾丽斯做下述工作：

1. 下载 (p, α, β)；

2. 选择随机数 k，并计算：$r \equiv \alpha^k \pmod p$；

3. 计算 $t \equiv \beta^k m \pmod p$；

4. 发送 (r, t) 给鲍勃。

鲍勃通过计算下式解密：

$$tr^{-a} \equiv m \pmod p。$$

这能起作用是因为

$$tr^{-a} \equiv \beta^k m(\alpha^k)^{-a} \equiv (\alpha^a)^k m\alpha^{-ak} \equiv m \pmod p。$$

如果伊芙知道了 a，那么她就可以使用和鲍勃相同的过程来解密。因此，鲍勃保证 a 始终都不被公开十分重要。数字 α，β 是公开的，且 $\beta \equiv \alpha^a \pmod p$，计算离散对数的困难就

在于如何保证 a 始终是安全的。

既然 k 是一个随机的整数，β^k就是一个随机的非零模 p 整数。因此，$t\equiv\beta^k m \pmod p$ 等于 m 乘以一个随机的非零整数，且 t 是一个模 p 随机数（除非 $m=0$，当然这种情况应该避免），所以 t 并没有给伊芙任何关于 m 的信息。即使知道了 r，也不能给伊芙足够的辅助信息。

由于这也是一个关于离散对数的问题，所以很难从 r 求得 k。但是，如果伊芙知道了 k，那么她就可以计算 $t\beta^{-k}$，结果就是 m。

每一个传送的信息都使用一个不同的随机数 k，这非常重要。假设艾丽斯加密信息 m_1和 m_2给鲍勃，鲍勃使用相同的 k。那么，对于这两个信息而言，r 是相同的，因此密文是 (r, t_1) 和 (r, t_2)。如果伊芙知道了 m_1，那么她也能够很快按如下过程求得 m_2。我们注意到

$$t_1/m_1\equiv\beta^k\equiv t_2/m_2 \pmod p,$$

既然伊芙知道了 t_1和 t_2，就能够根据 $m_2\equiv t_2 m_1/t_1 \pmod p$ 计算 m_2。

在第 15 章，我们会遇到与 ElGamal 算法类似的方法，该方法使用椭圆曲线。

7.5 习　　题

1. 设 $p=13$，计算 $L_2(3)$。

2. 设 $p=19$，那么 2 是本原根，用 Pohlig-Hellman 方法计算 $L_2(14)$。

3. (a) 设 α 是模 p 本原根，证明：

$$L_\alpha(\beta_1\beta_2)\equiv L_\alpha(\beta_1)+L_\alpha(\beta_2) \pmod{p-1}。$$

（提示：参见 3.7 节中的命题。）

(b) 更一般地，设 α 是一个随机数，证明：

$$L_\alpha(\beta_1\beta_2)\equiv L_\alpha(\beta_1)+L_\alpha(\beta_2) \pmod{\mathrm{ord}_p(\alpha)}。$$

其中，$\mathrm{ord}_p(\alpha)$ 的定义见习题 3.9。

4. (a) 若你有一个 500 位的随机素数 p。假设有人想要保存口令，将其写成数字的形式。如果 x 是口令，那么 $2^x \pmod p$ 被存储在一个文件中。当给定一个口令 y 时，$2^y \pmod p$ 与用户文件中的输入相比较。假设有人已经能够进入这个文件，为什么推测出口令仍然很困难？

(b) 假如所选择的 p 是一个 5 位的素数，为什么 (a) 中的系统是不安全的？

5. 用 Pohlig-Hellman 算法的观点，再次考虑一下练习 3.11 中的问题，惟一的素数 q 是 2。对 k 正如 3.11 一样，记 $k=x_0+2x_1+\cdots+2^{15}x_{15}$。

(a) 证明：用 Pohlig-Hellman 算法可以得到

$$x_0=x_1=\cdots=x_{10}=0$$

且

$$2=\beta=\beta_1=\cdots=\beta_{11}。$$

(b) 用 Pohlig-Hellman 算法计算 k。

7.6 上 机 题

1. 设 $p = 53047$，证明：$L_3(8576) = 1234$。

2. 设 $p = 3989$。

(a) 证明：$L_2(3925) = 2000$ 和 $L_2(1046) = 3000$。

(b) 计算 $L_2(3925 \cdot 1046)$。（注意：结果应该小于 3988。）

3. 设 $p = 31$，估计 $L_3(24)$ 的值。

4. 设 $p = 1201$，用 Pohlig-Hellman 算法计算 $L_{11}(2)$。

第 8 章 数字签名

很多年以来，人们用各种签名将他们的身份同文档联系起来。在中世纪，贵族用他们勋章的蜡印来封文档。这时假设贵族是惟一能够复制这种勋章的人。在现代事务中，通过划信用卡来签名。售货员应该通过与信用卡上的签名进行比较来检验签名。随着电子商务和电子文档的发展，这些方法已经不能满足需要了。

例如，假设你想签署一个电子文档。为什么不能将你的签名数字化并简单地附在文档上？因为任何得到该文档的人都可以简单地将你的签名移走并将其添加到其他地方，比如，大面额的支票。对于传统的签名，这需要将文档上的签名剪下来或者影印下来，然后将其粘贴到支票上，这很少能通过可信任的签名。然而这样的电子签名很容易伪造，并且很难与最初的区分。

因此，我们要求电子签名不能和消息分开并附加到其他消息上。也就是说，签名不仅仅和签名者联系，而且还与签名的消息联系在一起，同时签名需要很容易被其他方证实。因此数字签名包含两个不同的步骤：签名过程和认证过程。

下面，我们首先介绍两个签名方案，然后讨论散列函数的一些基本问题并且讨论数字签名的标准。我们还会讨论对于离散对数和数字签名方案的"生日"攻击。

8.1 RSA 签名

鲍勃有一个艾丽斯同意签署的文档，他们执行下面的步骤。

1. 艾丽斯生成两个大的素数 p，q，并且计算 $n=pq$。她选择满足 $1<e_A<\phi(n)$ 的 e_A 并且 $\gcd(e_A,\phi(n))=1$，计算 d_A 使得 $e_Ad_A\equiv 1(\bmod \phi(n))$。艾丽斯公布 (e_A,n) 并且秘密保存 d_A、p，q。

2. 艾丽斯的签名为

$$y\equiv m^{d_A}(\bmod\ n)。$$

3. 然后公开数对 (m,y)。

鲍勃能够通过下面的步骤验证艾丽斯真正签署了消息。

1. 下载艾丽斯的 (e_A,n)；

2. 计算 $z\equiv y^{e_A}(\bmod\ n)$。如果 $z=m$，那么鲍勃就认为消息是有效的，否则消息是无效的。

假设伊芙想将艾丽斯的签名附加到另一个消息 m_1 上。由于 $y^{e_A} \not\equiv m_1 \pmod n$，她不能简单地使用数对 (m_1, y)，因此她需要 y_1 使得 $y_1^{e_A} \equiv m_1 \pmod n$。这和解密 RSA 密文 m_1 得到明文 y_1 是同样的问题。这被公认为是很难做到的。

另外一种可能是伊芙首先选择 y_1，然后设消息 $m_1 \equiv y_1^{e_A} \pmod n$，在目前的方案中艾丽斯不能否认签署了消息 m_1，然而 m_1 不太可能是一个有意义的消息，它很可能是一些字母的随机序列，并且不会使她付给伊芙数百万美元。因此艾丽斯声称它是伪造的是不可信的。

在这个过程中还有一种可能是艾丽斯签署一个文档，而她不知道文档的内容。假设鲍勃有一个重要的发明，他想公开地记录他所做的工作（这样当申请诺贝尔奖的时候，他具有优先权）。但他并不想让其他任何人知道详细内容（如此他可从发明中获利）鲍勃和艾丽斯做下面的工作。待签名的消息是 m。

1. 艾丽斯选择一个 RSA 模数 $n(n = pq$，两个大素数的乘积$)$、一个加密指数 e 和解密指数 d。她公布 n 和 e，而秘密保留 p，q，d。事实上她可以在签名之后从内存中删除 p，q，d。

2. 鲍勃选择一个随机的整数 $k \pmod n$，其中 $\gcd(k,n) = 1$，并且计算 $t \equiv k^e m \pmod n$，他将 t 发送给艾丽斯。

3. 艾丽斯通过计算 $s \equiv t^d \pmod n$ 签署 t，并将 s 发送给鲍勃。

4. 鲍勃计算 $s/k \pmod n$，这就是签名后的消息 m^d。

假设 s/k 是签名的消息：注意到 $k^{ed} \equiv (k^e)^d \equiv k \pmod n$，因此这是简单的加密，然而解密则通过 RSA 方案中的 k。因此，

$$s/k \equiv t^d/k \equiv k^{ed}m^d/k \equiv m^d \pmod n$$

是签名后的消息。

k 的选择是随机的，$k^e \pmod n$ 是对一个随机数的 RSA 加密，因此也是随机的，所以 $k^e m \pmod n$ 给了 m 必要的无效信息（然而，这不能隐藏像 $m = 0$ 这样的信息）。这样艾丽斯就不知道她所签署的消息。

一旦签名完成，鲍勃就有了和通过标准的签名过程所得到的相同的消息。

这个协议还有一些潜在的危险。比如，鲍勃可以让艾丽斯签署一个付给他 100 万美金的协议。为了避免这些问题就需要安全认证，现在我们不讨论这些。

这种签名规程称为**盲目签名**（blind signatures），是由 David Chaum 发明的，他在这个问题上有很多专利。

8.2 ElGamal 签名方案

可以修改 7.4 节的 ElGamal 签名方案，从而提供一种新的签名方案，它与 RSA 方案的不同之处在于，对于 ElGamal 方法同一个消息有许多不同的有效签名。

假设艾丽斯想对一个消息签名。首先她选择一个大素数 p 和一个本原根 α，然后选择一个秘密的整数 a 使得 $1 \leqslant a \leqslant p-2$，并且计算 $\beta \equiv \alpha^a \pmod p$，$p$，$\alpha$，$\beta$ 的值是公开的，这个方案的安全性在于 a 的保密性。由于离散对数问题是非常困难的，对手很难由 (p, α, β) 确定 a。

艾丽斯为了签署一个消息 m，需要做下面的工作：

1. 选择一个安全的随机数 k，使得 $\gcd(k,p-1)=1$；
2. 计算 $r\equiv\alpha^k \pmod p$；
3. 计算 $s\equiv k^{-1}(m-ar) \pmod{p-1}$。

签署的消息是三元组 (m,r,s)。

鲍勃能够通过下面的步骤确认签名：

1. 下载艾丽斯的公钥 (p,α,β)；
2. 计算 $v_1\equiv\beta^r r^s \pmod p$ 和 $v_2\equiv\alpha^m \pmod p$；
3. 当且仅当 $v_1\equiv v_2 \pmod p$ 时签名是有效的。

现在我们来解释签名过程。假设签名是有效的，由于 $s\equiv k^{-1}(m-ar) \pmod{p-1}$，就有 $sk\equiv m-ar \pmod{p-1}$，于是 $m\equiv sk+ar \pmod{p-1}$。因此回忆一下用指数生成一个全部的模 p 同余式中模 $p-1$ 同余式的计算

$$v_2\equiv\alpha^m\equiv\alpha^{sk+ar}\equiv(\alpha^a)^r(\alpha^k)^s\equiv\beta^r r^s\equiv v_1 \pmod p。$$

假设伊芙发现了 a 的值，那么她能够处理签名过程，并且能够对任意文档实施艾丽斯的签名。因此，a 秘密地保存是非常重要的。

如果伊芙有另外一个消息 m，由于她不知道 a，因此她无法计算对应的 s。假设她试图通过选择一个满足验证等式的 s 来越过这一步，这意味着她需要满足

$$\beta^r r^s\equiv\alpha^m \pmod p$$

的 s。这个等式能够变换为 $r^s\equiv\beta^{-r}\alpha^m \pmod p$，这是一个离散对数问题。因此找到一个合适的 s 是很困难的，如果先选择 s，那么对于 r 的等式也类似于一个离散对数问题，并且更加复杂，一般认为这个问题也是很难解决的。尽管还不知道是否有一种方法能够同时确定 s 和 r，但这看上去是不可能的，因此只要模 p 离散对数的计算是困难的，那么这个签名方案就是安全的（例如 $p-1$ 不能是小素数的乘积，见 7.2 节）。

如果艾丽斯要签署另一个文档，她必须选择另外一个随机数 k。假设对于 m_1 和 m_2 她选择相同的 k，那么相同的 r 将用在两个签名中，因此伊芙会发现 k 被使用了两次。s 的值是不同的，称它们为 s_1 和 s_2。伊芙知道：

$$s_1k-m_1\equiv -ar\equiv s_2k-m_2 \pmod{p-1},$$

因此，

$$(s_1-s_2)k\equiv m_1-m_2 \pmod{p-1}。$$

设 $d=\gcd(s_1-s_2,p-1)$，该同余式存在合适的解 d，并且可以通过 3.3 节中介绍的方法得到。通常 d 很小，因此 k 可能的值没有很多，伊芙对每个可能的 k 计算 α^k 直到值 r，于是她知道了 k，现在伊芙解

$$ar\equiv m_1-ks_1 \pmod{p-1}$$

中的 a，这共有 $\gcd(r,p-1)$ 种可能。伊芙对于每一种可能计算 α^a 直到她获得 β，这样她也获得了对应的 a，这时她已经完全破解了这个体制并且可以随意地伪造艾丽斯的签名。

例：艾丽斯想签署消息 $m_1=151405$（这对应着消息 *one*，如果我们取 $01=a,02=b,\ldots$）。她选择 $p=225119$，那么 $\alpha=11$ 是本原根，她有一个秘密的数 a，并计算 $\beta=\alpha^a\equiv 18191 \pmod p$。为了签署这个消息，她选择一个随机数 k 并且秘密地保存，再计算 $r\equiv\alpha^k\equiv 164130 \pmod p$，然后她计算

$$s_1 \equiv k^{-1}(m_1 - ar) \equiv 130777(\text{mod } p-1),$$

签名后的消息是三元组（151405，164130，130777）。

现在假设艾丽斯又签署了消息 $m_2 = 202315$（对应着消息 *two*），并且得到签署后的消息为（202315，164130，164899），伊芙立即可以发现她选择了相同的 k，因此两个消息中 r 的值相同。因此她写下同余式

$$-34122k \equiv (s_1 - s_2)k \equiv m_1 - m_2 \equiv -50910(\text{mod } p-1)。$$

由于 $\gcd(-34122, p-1) = 2$，共有两个解，这可以通过 3.3 节中描述的方法得到。用 2 除同余式得到：

$$-17061k \equiv -25455(\text{mod } (p-1)/2),$$

这有解 $k \equiv 239(\text{mod}(p-1)/2)$，因此有两个 k（mod p）的值，即 239 和 $239 + (p-1)/2 = 112798$。计算

$$\alpha^{239} \equiv 164130, \alpha^{112798} \equiv 59924(\text{mod } p)。$$

由于第一个是 r 的正确值，伊芙计算 $k = 239$，她现在重写 $s_1 k \equiv m_1 - ar(\text{mod } p-1)$，得到

$$164130a \equiv ra \equiv m_1 - s_1 k \equiv 187104(\text{mod } p-1)。$$

由于 $\gcd(164130, p-1) = 2$，共有两个解，即 $a = 28862$ 和 $a = 141421$，这可以通过 3.3 节中的方法得到。伊芙计算

$$\alpha^{28862} \equiv 206928, \alpha^{141421} \equiv 18191(\text{mod } p)。$$

由于第二个值是 β，她将得到 $a = 141421$。

伊芙现在知道了 a，她可以在任何文档上伪造艾丽斯的签名。■

ElGamal 签名方案是**附加签名（signature with appendix）**的一个例子。由签名 (r,s) 恢复消息不是很容易，消息 m 必须包含在确认过程中，这是与 RSA 签名方案不同的地方，RSA 签名方案是一个**消息恢复方案（message recovery scheme）**。在这个实例中，消息直接从签名 y 中得到，因此，由于任何人都可以通过计算 $y^{e_A}(\text{mod } n)$ 得到 m，可见只需要传递 y，随机的 y 产生一个有意义的消息 m 是不可能的，所以有人试图通过替换 y 将一个有效的消息换成一个伪造的消息几乎没有什么危险。

8.3 散列函数

在前面讨论的两个签名方案中，签名至少和消息一样长，当消息很长时这是一个缺点。为了弥补这个问题，可以使用散列函数，签名方案被用在消息的散列值上而不是消息本身上。

一个**密码散列函数（cryptographic hash function）**h 是输入一个任意长度的消息并且输出固定长度的**消息摘要（message digest）**，正如图 8.1 中描述的 160 比特。对于函数 h，应该满足下面的属性：

1. 给定一个消息 m，消息摘要能够很快地计算出来。

2. 给定一个消息摘要，通过 $h(m) = y$ 得到 m 在计算上是不可行的（换句话说，h 是一个**单向的（one-way）**或**不可逆（preimage resistant）**的函数）。

3. 通过计算得到 m_1，m_2 使得 $h(m_1) = h(m_2)$ 是不可能的（这要求 h 是**很少有冲突的（strongly collisionfree）**）。

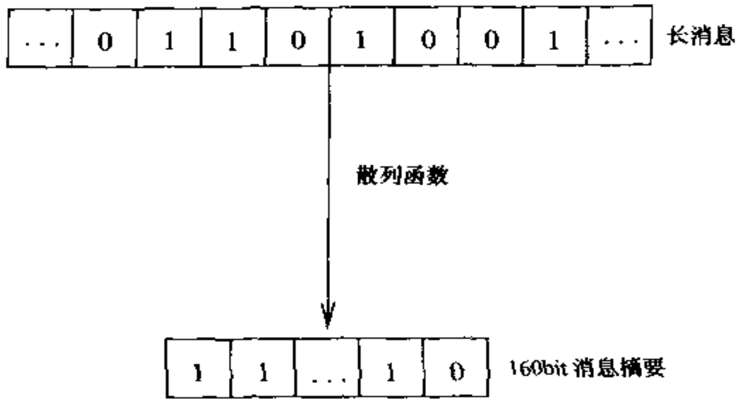


图 8.1 一个散列函数

由于可能的消息比可能的消息摘要多很多，可能经常会出现 m_1，m_2 使得 $h(m_1) = h(m_2)$ 的例子，属性 3 指出应该很难找到这样的实例。特殊情况下，如果鲍勃产生了一个消息 m 和它的散列 $h(m)$，艾丽斯想确定鲍勃不知道另一个消息 m' 使得 $h(m) = h(m')$，即使 m 和 m' 是随机的符号串。

例：假设 n 是一个大整数。设 $h(m) = m \pmod n$ 是介于 0 到 $n-1$ 之间的一个整数，这个函数很明显满足属性（1）。然而不满足（2）和（3）：给定 y，设 $m = y$，那么 $h(m) = y$，因此 h 不是单向函数。同样，选择任意两个对 n 取模余数值相同的 m_1 和 m_2，那么 $h(m_1) = h(m_2)$，因此 h 不是很少有冲突的。 ■

例：下面的例子有时候被称为离散对数散列函数，要归功于 Chaum，van Heijst 和 Pfitzmann。它满足属性（2）和属性（3），但是由于运算太慢，很难在实践中应用，然而它展示了散列函数的基本思想。

首先我们选择一个大素数 p，使得 $q = (p-1)/2$ 也是一个素数（见练习 8），我们选择素数 p 的两个本原根 α 和 β，由于 α 是一个本原根，存在 a 使得 $\alpha^a \equiv \beta \pmod p$。然而，我们假设并不知道 a（如果 a 预先不给出，寻找 a 是一个离散对数问题，这被认为很难）。

散列函数 h 将把整数 mod q^2 映射到整数 mod p，因此消息摘要包含了大约一半的原始比特信息，这和实践中要求的消息摘要的大小有相当大的减少不一致，但是这符合我们的目的。

记 $m = x_0 + x_1 q$，其中 $0 \le x_0, x_1 \le q-1$，那么

$$h(m) \equiv \alpha^{x_0}\beta^{x_1} \pmod p。$$

下面证实 h 是很少有冲突的。

命题：如果知道消息 $m \ne m'$ 并且 $h(m) = h(m')$，那么我们能够确定离散对数 $a = L_\alpha(\beta)$。

证明：记 $m = x_0 + x_1 q$，$m' = x'_0 + x'_1 q$，假设

$$\alpha^{x_0}\beta^{x_1} \equiv \alpha^{x'_0}\beta^{x'_1} \pmod p,$$

利用 $\beta \equiv \alpha^a \pmod p$，我们可以重写上式为

$$\alpha^{a(x_1 - x'_1) - (x'_0 - x_0)} \equiv 1 \pmod p。$$

由于 α 是一个模 p 本原根，我们知道当且仅当 $k \equiv 0 \pmod{p-1}$ 时 $\alpha^k \equiv 1 \pmod p$。在本例

中，这意味着

$$a(x_1 - x_1') \equiv x_0' - x_0 \pmod{p-1}。$$

取 $d = \gcd(x_1 - x_1', p-1)$，显然有 d 个解满足前面的同余式（见 3.3 节），并且 d 个解能够很快地找到。对于选定的 p，$p-1$ 的所有因子为 1，2，q，$p-1$，由于 $0 \leq x_1, x_1' \leq q-1$，可以推得 $-(q-1) \leq x_1 - x_1' \leq q-1$。因此，如果 $x_1 - x_1' \neq 0$，那么 $x_1 - x_1'$ 是 d 的非零倍数并且绝对值小于 q。这意味着 $d \neq q$，$p-1$，因此 $d=1$ 或 2，可知 a 最多有两种可能性，对于每种可能性计算 α^a，只有其中一个能够产生 β，因此得到了 a，正如我们所希望的那样。

另一方面，如果 $x_1 - x_1' = 0$，那么上面的过程产生 $x_0 - x_0' \equiv 0 \pmod{p-1}$，因为 $-(q-1) \leq x_0' - x_0 \leq q-1$，我们必须有 $x_0' = x_0$，因此 $m \neq m'$，与我们的假设相反。 □

现在我们假设有一个函数 g 能够由一个消息摘要 y 很快得到一个消息 m 使得 $h(m) = y$。在这个例子中很容易找到 $m_1 \neq m_2$ 满足 $h(m_1) = h(m_2)$：选择一个随机数 m，计算 $y = h(m)$，然后计算 $g(y)$，由于 h 把 q^2 个消息映射到 $p-1 = 2q$ 个消息摘要上，所以存在许多消息 m' 使得 $h(m') = h(m)$，因此 $m' = m$ 的可能性不大。如果 $m' = m$，那么可以选择另外一个随机数 m。很快我们能够发现一个发生抵触的消息，即 $m' = m$，但是 $h(m_1) = h(m_2)$，前面的命题显示我们能够解决一个离散对数问题，因此不可能存在这样一个函数 g。

正如前面我们所提到的，这个散列函数很适用于概念介绍，但是由于它运算慢的特性，并不适合实际应用，尽管通过循环平方法能够使计算效率提高，但是即使是使用循环平方法，对实际应用来说也太慢，在诸如电子商务等应用中，软件中执行乘法还需求额外的时间是不允许的。现在有许多专业的、高效的散列函数，例如 Rivest 发现的 MD 系列，具体而言 MD4 以及它的增强版 MD5 被广泛地采用，并且可对任意长度的消息产生 128 比特的消息摘要。另一个选择是 NIST 的安全散列运算法则（SHA），它可以产生 160 比特的消息摘要。在这里描述这些运算法则没有很大的作用，有关这些内容以及其他散列函数的细节，请参见 [Stinson]，[Schneier] 和 [Menezes et al.]。

散列、签名和应用

散列函数的一个有效的应用是使签名方案更有效。散列函数是公开的，对于一个消息 m，艾丽斯计算它的散列值 $h(m)$，$h(m)$ 的结果非常小，因而对散列的签名要比对所有消息的签名快得多，艾丽斯对每个散列函数计算签名消息 $\text{sig}(h(m))$，并将其作为消息的签名，数对 $(m, \text{sig}(h(m)))$ 传达了和最初的签名方案相同的信息。这种方案的优势是创建签名速度快（在散列操作非常快这个合理的前提之下），并且传输和存储需要的资源也更少。

这种方法安全吗？假设伊芙占有了艾丽斯的签名 $(m, \text{sig}(h(m)))$，她有另外一个消息 m' 想加到艾丽斯的签名上，这意味着她需要 $\text{sig}(h(m')) = \text{sig}(h(m))$；具体而言她需要 $h(m') = h(m)$。如果散列函数是单向的，那么伊芙将发现很难找到这样的一个 m'，她所期待的 m' 满足 $h(m') = h(m)$ 的机会很小。而且，由于我们要求所用的散列函数很少有冲突，因此伊芙找到两个签名相同的消息 $m_1 \neq m_2$ 是不太可能的。当然，如果她能够找到，那么她就能够得到艾丽斯的签名 m_1，然后把她的签名转移到 m_2 上。但是，由于 m_1（和 m_2）很可能是无意义的消息，因此艾丽斯将会怀疑这个签名。

下一节我们将看到当消息摘要太小时伊芙能够欺骗艾丽斯（并且我们也会看到散列函数并不是很少有冲突的）。

散列函数也可以用作数据完整性校验，数据完整性问题来自两种基本的情况，一种情况是当数据（加密或不加密）传输给另外一个人时，噪声信道向数据中引入了错误；另一种情况是观察器在接收者得到数据之前重排了数据传输的某些方式。每一种原因都会使数据被破坏。

例如，假设艾丽斯要与鲍勃传送与伊芙有关的很长的财务事项信息并且分组加密，也许伊芙推测出每个消息列的第10组是存到伊芙账户上的钱的数量，她将很容易用另外一个消息代替第10组数据以增加存款。

另外一种情况下，艾丽斯传送给鲍勃由好几个数据分组组成的消息，但是其中一组在传输中丢失，鲍勃可能不知道这一分组丢失了。

这里就是散列函数可以起作用的地方了。假设我们通过一个通信信道发送消息 $(m, h(m))$ 并且收到 (M, H)，为了检查是否有错误发生，接收者计算 $h(M)$ 看结果是否等于 H。如果有任何错误发生，由于散列函数具有很少冲突的性质，很可能 $h(M) \neq H$。

8.4 生日攻击

如果一个房间里有23个人，两个人生日相同的概率略大于50%，如果有30个人，那么概率大概是70%，这也许看起来很奇怪，这被称为**生日判定**（birthday paradox）。我们看看为什么会这样，这里忽略闰年并且假设所有人的生日有相同的可能性（如果不是这样，给出的概率可能会高一点）。

考虑23个人的情况，计算他们有不同生日的概率。我们把他们排成一行，第一个人的生日占了一天，因此第二个人有不同生日的概率是（1－1/365），第三个人少了两天，因此他和前两个人生日不同的概率是（1－2/365），因此三个人有不同生日的概率是（1－1/365）（1－2/365），以此类推我们得到23个人生日各不相同的概率是

$$\left(1-\frac{1}{365}\right)\left(1-\frac{2}{365}\right)\cdots\left(1-\frac{22}{365}\right)=.493$$

因此，至少有两个人生日相同的概率是

$$1-.493=.507$$

理解前面的推理的一个直观的方法是考虑40个人的情况，如果前30个人满足我们所做的假设，即假设30个人有不同的生日，现在我们需要选择后10个生日，由于30个生日已经选择了，随机选择一个人的生日与头30个人中的一个生日相同的概率大概是10%，并且我们选择10个生日，因此我们得到一个与前面生日相同并不奇怪。事实上40个人中有两个人生日相同的概率大约是89%。

更一般的情况，假设我们有 n 个对象，其中 n 很大，有 r 个人，并且每个人选择一个对象（可以置换，因此几个人可以选择相同的对象）。如果 $r \approx 1.177\sqrt{n}$，那么至少有两个人选择相同对象的概率大概是50%。如果 $r \approx \sqrt{2\lambda n}$，那么相同的概率是 $1-e^{-\lambda}$（注意这只是对于很大的 n 的一个近似；对于较小的 n 采用前面的方法得到准确的结果要更好）。

这些思想用在密码学上需要一些小的调整。假设有两个房间，每个房间有30人，那么第一个房间的某个人和第二个房间的某个人有相同生日的概率是多少？更一般的情况，假设

有 n 个对象，两组各有 r 个人，每组中的每个人选择一个对象（可以置换），那么第一组中某个人选择的对象与第二组中某个人选择的对象相同的概率是多少？同样，如果 $r \approx \sqrt{\lambda n}$，那么两个人选择相同对象的概率是 $1 - e^{-\lambda}$，i 个人选择相同对象的概率是 $\lambda^i e^{-\lambda}/i!$，这个问题最普遍的分析在[Girault et al.]中给出。

8.4.1　签名方案中的生日攻击

艾丽斯要通过某种签名方案对文档的散列值签名来签署一个电子文档，假设散列函数产生一个 50 比特的输出，她担心佛瑞德(Fred)哄骗她签署另外一个合同，也许是关于佛罗里达的沼泽地。由于欺骗性的合同和正确的文档有相同散列值的概率是 $1/2^{50}$，这大概是 $1/10^{15}$，因此艾丽斯感觉是安全的。佛瑞德可以尝试许多欺骗性的合同，但是他找到有相同散列值的合同的可能性非常小，但是佛瑞德研究了生日问题并且照着下面的方法去做，他找到了能够对文档进行细微改变的 30 个位置：在一行的末尾增加一个空格，略微改变一个词的拼写，等等。在每一个位置他有两个选择，要么做一个细微的改变要么保留原状，因此他能够产生 2^{30} 个本质上与原始文档相同的文档，同样他得到 2^{30} 个欺骗性的合同并且存储它们的散列值。考虑最初的生日问题，其中 $r = 2^{30}$ 并且 $n = 2^{50}$，我们有 $r = \sqrt{\lambda n}$，其中 $\lambda = 2^{10} = 1024$。因此一个好文档的版本和一个欺骗性的合同有相同散列值的概率是 $1 - e^{-1024} \approx 1$。佛瑞德找到这一对匹配的文档，并且让艾丽斯签署其中好的文档版本，他计划将她的签名附加在欺骗性的合同之上，由于它们有相同的散列值，因此签名对于欺骗性的合同将会有效，所以佛瑞德会声称艾丽斯同意购买沼泽地。然而艾丽斯是一个英语教师，她坚持从一个句子中删除一个逗号，这样就和佛瑞德要求她签名的文档有着完全不同的散列值，佛瑞德又被挫败了，这时他面临的问题是找到一个欺骗性的合同和新的正确文档具有相同的散列值，这根本是不可能的。

佛瑞德的行为被称为生日攻击，由于生日攻击有效地将比特数对分，在实际应用中只要你认为有必要，就可以对输出使用两次散列函数。对于签名方案的生日攻击，艾丽斯所做的是一种可取的方法，在签名电子文档时要做一个小的改动。

8.4.2　基于离散对数的生日攻击

假设我们处理一个大的素数 p，并且想要计算 $L_\alpha(\beta)$，换句话说，我们想要解 $\alpha^x \equiv \beta \pmod p$，通过生日攻击，我们很可能做到。

构造两个列表，长度都大概是 $\sqrt{p}$：

1. 对于随机选择的近似于 $\sqrt{p}$ 的 k 值，第一个列表包含数字 $\alpha^k \pmod p$。

2. 对于随机选择的近似于 $\sqrt{p}$ 的 l 值，第二个列表包含数字 $\beta\alpha^{-l} \pmod p$。

有可能第一个列表的某个元素和第二个列表的某个元素相匹配。如果这样，我们有

$$\alpha^k \equiv \beta\alpha^{-l}，因此\ \alpha^{k+l} \equiv \beta \pmod p。$$

因此，$x \equiv k + l \pmod{p-1}$ 是期望的离散对数。

8.4.3　双重加密的中间相遇攻击

下面讲述的问题与生日攻击类似，但是该攻击中寻找匹配消息的方法是确定问题而不是概率问题。这就意味着可以确保找到一个匹配消息，而不是仅仅存在一个很大的可能性。但是，第二个步骤，也就是去掉额外的匹配信息，却是个概率问题。

艾丽斯和鲍勃使用一种加密方法，加密函数记作 E_k，解密函数记作 D_k，其中 k 是密钥。我们假设如果某人知道 k，那么他也知道 E_k 和 D_k（因此艾丽斯和鲍勃能够使用经典的、非公开的密钥体制）。他们有一个好主意，不是加密一次而是使用 k_1,k_2 加密两次。以明文消息 m 开始，密文是 $c=E_{k2}(E_{k1}(m))$，为了解密只需要计算 $m=D_{k1}(D_{k2}(c))$，伊芙需要得到 k_1,k_2 来解密他们的消息。

这是否能够提供更好的安全性呢？对于许多加密体制，加密两次等价于用其他的密钥加密一次，例如，两个仿射函数的合成仍然是一个仿射函数（见练习2.5）。同样，用指数 e_1 和 e_2 使用两次 RSA 加密（使用相同的 n）对应于用指数 e_1e_2 进行一次加密，在这些例子中两次加密并没有优点。然而还有些体制比如 DES（见4.4节）两次加密的组合不等于简单地用另外一个密钥加密，对于这些体制来说，两次加密可能提供更高的加密强度，但下面的攻击显示，只要我们有一台内存足够大的计算机，两次加密也不能提供更高的加密强度。

假设伊芙截获了一个消息 m 和经过两次加密的密文 $c=E_{k2}(E_{k1}(m))$，她想找到 k_1 和 k_2。她首先对所有可能的密钥 k 计算并存储 $E_k(m)$，然后对所有可能的密钥 k 计算 $D_k(c)$，最后比较两个列表，由于其中必有一个正确的密钥对，所以与生日攻击相比，除了她知道至少有一个匹配以外，其他的和生日攻击类似。如果有几对都能匹配，那么她选择其他的明文密文对来确定她得到的哪一对能够正确地匹配明文和密文，这能够很大地减少列表数量，如果列表中剩下的多于一对，那么她继续上面的操作直到剩余一对匹配（或者她确定有两个或更多的密钥对给出双重加密功能），现在伊芙有了期望的密钥对 k_1,k_2。

如果伊芙只有一个明文密文对，她仍然能将可能的密钥对列表减少至较少的数量，如果她截获了另外一个传输，那么她能够尝试每一种可能，并且得到一个很短的有意义的明文列表。

如果有 N 个可能的密钥，伊芙需要计算并存储 N 个 $D_k(m)$ 值，然后她需要计算另外 N 个 $D_k(c)$ 值并且将它们与存储列表作比较，但是这 $2N$ 个计算（加上比较）比起在所有的密钥对 k_1,k_2 中寻找所需要的 N^2 个计算要少得多。

中间相遇过程比穷尽搜索一次加密的所有密钥花费的时间要略多一些，并且需要许多内存来存储第一个列表，然而结论是双重加密并不能在所有的场合大大地提高安全级别。

同样，我们能够使用三重加密，使用三个密钥。一个类似的攻击使三重加密降低到最多为人们对双重加密最初所期待的强度，即可能的密钥数平方。

8.5　数字签名算法

美国全国科学技术学会在1991年提出了数字签名算法（DSA），并在1994年将它作为标准。正如 Elgamal 方法一样，DSA 是一个有附加数字签名方案，同时，和其他方案一样，通常

是一个签名的消息摘要，在这种情况下散列函数产生了一个 160 比特的输出。下面的叙述中假设我们的数字消息 m 已经被散列函数处理过，因此我们要签名的是一个 160 比特的消息。

DSA 的密钥按照下面的步骤产生，首先有一个初使化的过程：

1. 艾丽斯找到一个 160 比特的素数 q 和一个满足 $q \mid p-1$ 的素数 p（见练习 8）。离散对数问题会使 p 的选择很困难（在算法的最初版本中 p 有 512 比特，在算法的以后版本中允许有更长的素数）。
2. 设 g 是一个模 p 的本原根并且 $\alpha \equiv g^{(p-1)/q} \pmod p$，那么 $\alpha^q \equiv 1 \pmod p$。
3. 艾丽斯选择了一个保密的 a，使其满足 $1 \leq a < q-1$，并且计算 $\beta \equiv \alpha^a \pmod p$。
4. 艾丽斯公布 (p, q, α, β) 并且秘密地保存 a。

艾丽斯通过下面的过程签署一个消息 m：

1. 选择一个随机的秘密的整数 k 满足 $0 < k < q-1$。
2. 计算 $r = (\alpha^k \pmod p) \pmod q$。
3. 计算 $s \equiv k^{-1}(m + ar) \pmod q$。
4. 艾丽斯对 m 的签名是 (r, s)，签名将和 m 一起发送给鲍勃。

鲍勃为了验证签名，他必须：

1. 下载艾丽斯的公开信息 (p, q, α, β)。
2. 计算 $u_1 \equiv s^{-1}m \pmod q$ 和 $u_2 \equiv s^{-1}r \pmod q$。
3. 计算 $v = (\alpha^{u_1}\beta^{u_2} \pmod p) \pmod q$。
4. 当且仅当 $v = r$ 时接受签名。

我们证明这个验证工作。由 s 的定义，有

$$m \equiv (-ar + ks) \pmod q,$$

这隐含着

$$s^{-1}m \equiv (-ars^{-1} + k) \pmod q。$$

因此，

$$\begin{aligned} k &\equiv s^{-1}m + ars^{-1} \pmod q \\ &\equiv u_1 + au_2 \pmod q。\end{aligned}$$

于是 $\alpha^k = \alpha^{u_1 + au_2} = (\alpha^{u_1}\beta^{u_2} \pmod p) \pmod q$，这样 $v = r$。

和 ElGamal 方案一样，整数 a 必须秘密地保存，任何人只要知道 a 就可以签署期望的任意文档。同样，如果相同的 k 值使用了两次，通过前面的过程找到一个 a 是可能的。

和 ElGamal 方案相比，整数 r 并没有包含所有的密钥信息，知道 r 只是允许我们找到模 q 的值，从大约 $2^{512-160} = 2^{342}$ 个模 p 数减少为给定的模 q 数。

使用 $\alpha^q \equiv 1 \pmod p$ 比使用一个本原根有什么好处呢？回忆一下解决离散对数问题的 Pohlig-Hellman 攻击，可以找到模小的素数因子 $p-1$ 的信息，而对模大的素数因子比如 q 是无效的。在 ElGamal 方案中，一个攻击者可以确定 $a \pmod{2^t}$，其中 2^t 是 2 除 $p-1$ 的最大的幂，这对于找到 a 没有什么益处，但是收集小的信息块通常是很有用的，DSA 方案通过移除 a 的模 q 信息以外的所有信息避免了这个问题的发生。

在 ElGamal 方案中，验证的步骤中需要 3 个模的幂运算，这个步骤在 DSA 方案中进行了修改，因此仅仅需要两个模的幂运算。由于模的幂运算是运算中很慢的一个部分，所以这个改变使验证过程加快。如果在短时间内有很多签名需要验证，这个改变是很重要的。

8.6 习　题

1. 证明：如果某个人发现了 ElGamal 签名方案中 k 的值，那么也就可以确定 a 了。

2. 假设 (x, γ, δ) 是一个用 ElGamal 签名方案签名的消息，选择 h 满足 $\gcd(h, p-1) = 1$，并且设 $\gamma_1 \equiv \gamma^h \pmod p$，$\delta_1 \equiv \delta\gamma_1 h^{-1}\gamma^{-1} \pmod{p-1}$。

(a) 寻找一个消息 x_1，使得 $(x_1, \gamma_1, \delta_1)$ 是一个有效的签名。

(b) 这种方法使得伊芙能够伪造消息 x_1 的签名，为什么这不大可能产生问题呢？

3. 设 $p = 11$，$q = 5$，$\alpha = 3$，并且 $k = 3$。证明 $(\alpha^k \pmod p) \pmod q \neq (\alpha^k \pmod q) \pmod p$，这表明 DSA 中操作的顺序是很重要的。

4. 设 p 是一个素数并且 α 满足 $p \nmid \alpha$，设 $h(x) \equiv \alpha^x \pmod p$。解释 $h(x)$ 为什么不是一个好的密码散列函数。

5. 通过改变 ElGamal 数字签名方案的签名等式 $s \equiv k^{-1}(m - ar) \pmod{p-1}$，可以得到许多这种签名方案的变形，下面是一些变形。

(a) 考虑签名等式 $s \equiv a^{-1}(m - kr) \pmod{p-1}$，证明：验证等式 $\alpha^m \equiv (\alpha^a)^s r^r \pmod p$ 是有效的验证过程。

(b) 考虑签名等式 $s \equiv am + kr \pmod{p-1}$，证明：验证等式 $\alpha^s \equiv (\alpha^a)^m r^r \pmod p$ 是有效的验证过程。

(c) 考虑签名等式 $s \equiv ar + km \pmod{p-1}$，证明：验证等式 $\alpha^s \equiv (\alpha^a)^r r^m \pmod p$ 是有效的验证过程。

6. ElGamal 签名方案对于一种称为*存在伪造*的攻击是脆弱的，这里是基本的存在伪造攻击。选择 u，v 使得 $\gcd(v, p-1) = 1$，计算 $r \equiv \beta^v \alpha^u \pmod p$ 并且 $s \equiv -rv^{-1} \bmod (p-1)$。

(a) 证明 (r, s) 数对是消息 $m = su \pmod{p-1}$ 的一个有效的签名（当然 m 很可能不是一个有意义的消息）。

(b) 假设使用散列函数 h，并且签名必须对 $h(m)$ 而不是对 m 有效（因此我们需要有 $h(m) = su$）。解释为什么这个方案能够抵御存在伪造攻击，也就是说为什么通过这种过程签名的消息很难产生一个伪造。

7. 艾丽斯想要通过 ElGamal 签名方案签署一个文档，假设她的随机数产生器坏了，在签名方案中她使用 $k = a$，伊芙怎样能够发现这一点？并且怎样确定 k 和 a 的值（从而破坏这个体制）？

8. (a) 在许多加密协议中，需要选择一个素数 p 满足 $q = (p-1)/2$ 也是素数。一种方法是随机地选择一个素数 q 并且测试 $2q+1$ 是否为素数，假设选择的 q 大约有 100 位十进制数，假定 $2q+1$ 是一个 100 位的随机的奇数（这并不十分准确，由于 $2q+1$ 不能同余于 1 模 3。但是这个假设对于粗略的估计是足够的了）。证明 $2q+1$ 是素数的可能性大概是 1/115（使用素数理论，如 6.3 节所述）。这意味着对每 115 个随机素数 q 的选择，你可以找到一个合适的素数 p。

(b) 在数字签名算法的一个方案中，艾丽斯需要一个 160 比特的素数 q 和一个 512 比特的素数 p，满足 $q \mid p-1$。假设艾丽斯选择了一个随机的 160 比特的素数 p 和一个满足 $qk+1$

有512比特的随机的 352 比特的偶数 k。证明 $qk+1$ 是素数的可能性大约是 1/177，这意味着艾丽斯能够很快地找到一个合适的 q 和 p。

9. 考虑下面 ElGamal 签名方案的变形。艾丽斯选择一个大的素数 p 和一个本原根 α，对于给定的整数 x 满足 $0 \le x < p$，艾丽斯选择一个函数 $f(x)$，返回一个整数 $f(x)$ 满足 $0 \le f(x) < p-1$（例如，对于 $0 \le x < p$，$f(x) = x^7 - 3x + 2 \pmod{p-1}$ 就是这样一个函数）。她选择一个秘密的整数 a 并且计算 $\beta \equiv \alpha^a \pmod p$，数字 p，α，β 和函数 $f(x)$ 是公开的。

艾丽斯想要签署一个消息 m：

1. 艾丽斯选择一个随机整数 k 满足 $\gcd(k, p-1) = 1$。
2. 她计算 $r \equiv \alpha^k \pmod p$。
3. 她计算 $s \equiv k^{-1}(m - f(r)a) \pmod{p-1}$。

签名的消息是 (m, r, s)。

鲍勃按下述过程验证签名：

1. 他计算 $v_1 \equiv \beta^{f(r)} r^s \pmod p$。
2. 他计算 $v_2 \equiv \alpha^m \pmod p$。
3. 如果 $v_1 \equiv v_2 \pmod p$，表示签名是有效的。

(a) 证明如果所有的乘积都正确，那么验证等式正确。

(b) 假设艾丽斯很懒，对于所有的 x 都选择常数函数满足 $f(x) = 0$。证明伊芙能够对每个消息 m_1 伪造一个签名（例如，给定一个值 k，r 和 s 将给出消息 m_1 的一个有效签名）。

10. 设 E_K 和 D_K 描述一个加密体制的加密和解密，其密钥为 K，因此如果 m 是明文，那么 $E_K(m) = c$ 是密文并且 $D_K(c) = m$，假设如果 K 和 L 是两个密钥，那么很容易找到（并且存在）一个密钥 K_1，对于所有的 m 都满足 $E_L(E_K(m)) = E_{K1}(m)$。假设已知一个明文密文对 (m_0, c_0)。

(a) 说明怎样使用一个生日攻击得到密钥 K_0，使得 $E_{K_0}(m_0) = c_0$。

(b) 假设 K_1 和前面一样，存在但是很难找到。说明怎样利用中间相遇攻击找到一个解密函数（但是你可能找不到解密密钥）。

8.7 上 机 题

1. 假设我们采用 ElGamal 签名方案，其中 $p = 65539$，$\alpha = 2$，$\beta = 33384$，我们发送两个签名的消息 (m, r, s)：

$$(809, 18357, 1042)\ (= \text{hi})\ 和$$

$$(22505, 18357, 26272)\ (= \text{bye})。$$

(a) 证明每个签名使用相同的 k 值。

(b) 使用 (a) 中的结论找到这个 k 值，并且找到一个 a 使得 $\beta \equiv \alpha^a \pmod p$。

2. （如果没有 Maple Kernel 的帮助，这个问题中的数据对于 MATLAB 来说太大了。）艾丽斯和鲍勃有下面的 RSA 参数：

$n_A = 171024704183616109700818066925197841516671277$，$e_A = 1571$

$n_B = 839073542734369359260871355939062622747633109$，$e_B = 87697$。

鲍勃知道

$$p_B = 987634576978345689346 13,\quad q_B = 8495789457893457345793$$

（其中 $n_B = p_B q_B$）。艾丽斯签署了一个文档并且把这个签名（m, s）（其中 $s \equiv m^{d_A} \pmod{n_A}$）发给了鲍勃，为了使文档的内容保密，她用鲍勃的公钥对文档加密，鲍勃收到加密的签名对 $(m_1, s_1) \equiv (m^{e_B}, s^{e_B}) \pmod{n_B}$，其中

$$m_1 = 14823765232498712344512418717130930$$

$$s_1 = 43176121628465441340112418672065063。$$

找到消息 m 并且验证消息来自艾丽斯。（签名对存为 *sigpairm*1，*sigpairms*1。数字 n_A，n_B，p_A，p_B 存储为 *signa*，*signb*，*sigpa*，*sigpb*。）

3.（如果没有 Maple Kernel 的帮助，这个问题中的数据对于 MATLAB 来说太大了。）在问题 2 中，假设鲍勃有素数 $p_B = 7865712896579$ 和 $q_B = 8495789457893457345793$，假定采用相同的加密指数，解释当艾丽斯发给鲍勃消息对（m_2，s_2）时他为什么不能验证她的签名。其中

$$m_2 = 14823765232498712344512418717130930,$$

$$s_2 = 43176121628465441340112418672065063。$$

需要什么更改才能使这个过程工作？（签名对存储为 *sigpairm2*，*sigpairs2*。）

4. (a) 如果一个教室里有 30 个人，那么至少两个人有相同生日的概率是多少？

(b) 教室中有多少人才能使最少有两个人生日相同的概率是 99%？

(c) 教室中有多少人才能使最少有两个人生日相同的概率是 100%？

5. 在一个四口之家，没有两个人的生日在同一个月的概率是多少？（假设所有的月份有相同的概率。）

6. 一个教授用每个学生社会保障号的后四位为一个班级的学生评分。在一个 200 人的班级中，至少有两个学生有相同的四位数的概率是多少？

第 9 章 电子商务与数字现金

随着通信技术的发展，如因特网和无线上网，产生了新型的贸易方式，这种可以触及更多顾客的潜在方式给偷窃和欺诈也带来了极大的可能性，在没有保护的通道中传送信用卡和信息交易可能会引起非法顾客的进入以及盗窃至关重要的信用信息，因此保护电子商务中必要信息的安全变得非常重要。

本章中我们要看两个如何在电子商务中进行加密的事例。第一个例子是安全电子传输™（SET）协议，它为电子传输中信用卡信息的交换提供了标准化，除了提供保护和隐私外，它还致力于阻止交替性的购买和信用信息的伪造；第二个例子是一个数字现金的模型，一种可以与钞票相媲美的数字记录，当使用银币和纸钞进行交易时，顾客可以放心他或她的身份对卖主是不公开的。在电子世界中，文档代替了硬币，交换它即可支付产品和服务。一个目标是可匿名，因为文档很容易被复制，如果我们保证匿名，那么必须有防止伪造的手段。在 9.2 节中，我们将展示如何在数字现金系统中达到上述两个目标。

9.1 安全的电子交易

每当有人在因特网上下达一个电子交易的命令时，大量的信息流就会传送，这些数据必须得到保护，防止被不友好地偷看，以保证顾客的隐私和防止信用欺骗。一个好的电子商务系统应包括以下几点：

1. **真实性（Authenticity）**：交易的参与者不可能模仿或者伪造签名。
2. **完整性（Integrity）**：有些类似于购买命令或支付说明的文件不能更改。
3. **私有性（Privacy）**：交易的细节应该尽可能保证安全。
4. **安全性（Security）**：敏感的帐目信息例如信用卡号必须受到保护。

所有这些要求都必须满足，哪怕是在公共的传输通道中，例如因特网上。

1996 年，信用卡公司 MasterCard 和 Visa 号召建立电子商务的标准，结果是它的发展又涉及到了更多的公司，称之为 SET，或者称为安全电子传输™。它起始于已经存在的信用卡系统，并允许顾客在开放的系统中安全地使用它。

SET 协议相当复杂，涉及到例如保证由可靠的权威机构来进行身份验证，以确保持卡者和销售商的合法性，以及确定使用何种支付方式，下面我们将讨论整个协议中的一个方面，即双重签名的使用。

下文中有几种可能的变化。例如，为了改进速度，可以将快速的对称密钥体制同公钥体

制结合使用，如果需要传输大量的信息，一个随机选定的对称密钥与一个长信息的散列值可以通过公钥体制来发送，这里的长信息本身是通过更快的对称密钥体制来发送的。然而，我们将把注意力限制到仅使用公钥方法的最简单的情况。

假设艾丽斯想买一本书名叫《如何用别人的信用卡号来诈骗银行》（*How to Use Other People's Credit Card Numbers to Defraud Banks*）的书，她曾经在网上看到过这本书的广告，很明显，如果将她的信用卡信息给出版者，她会感觉很不舒服，而且她不希望发给她信用卡的银行知道她在买什么。很多的交易都有类似的情况，银行没有必要知道顾客在干什么，而且为了安全起见，商人不应该知道信用卡号。但是，这两种信息又应以某种方式连接在一起，否则，消费可能会利用别人的定单，**双重签名（Dual signatures）**解决了这个问题。

以下的三方参与者是持卡者（即购买者）、销售商以及银行（信用卡使用授权者）。

持卡者有两条信息：

- GSO = 商品及服务要求（Goods and Services Order），包括持卡者和销售商的姓名、每一项定购的数量、价格，等等。
- PI = 支付说明（Payment Instructions），包括销售商的姓名、信用卡号、总价格，等等。

这个体制采用了公开的散列函数，称之为 H，同时用到了一个诸如 RSA 的公钥密码体制以及持卡人和银行各自拥有的公钥和私钥。令 E_C，E_M，E_B 分别表示持卡人、销售商和银行的（公开）加密函数，D_C，D_M，D_B 表示相应的（私有）解密函数。

持卡人执行如下步骤：

1. 计算 $GSOMD = H(E_M(GSO))$，它是 GSO 加密的消息摘要或散列函数。
2. 计算 $PIMD = H(E_B(PI))$，这是一个 PI 加密的消息摘要。
3. 连接 $GSOMD$ 和 $PIMD$，得到 $PIMD \parallel GSOMD$，然后算出其结果的散列函数，以得到支付定购的消息摘要 $POMD = H(PIMD \parallel GSOMD)$。
4. 通过计算 $DS = D_C(POMD)$ 签署 $POMD$，这是一个双重签名。
5. 发送 E_M（GSO），DS，$PIMD$ 和 E_B（PI）给销售商。

然后，销售商做以下事情：

1. 计算 $H(E_M(GSO))$（应该等于 $GSOMD$）。
2. 计算 $H(PIMD \parallel H(E_M(GSO)))$ 和 $E_C(DS)$。如果它们相等，那么销售商就证实了持卡人的签名，由此确信是持卡人的定购。
3. 计算 $D_M(E_M(GSO))$ 来获得 GSO。
4. 将 $GSOMD$，E_B（PI），DS 传送给银行。

现在银行要执行如下步骤：

1. 计算 $H(E_B(PI))$（应该等于 $PIMD$）。
2. 连接 $H(E_B(PI))$ 和 $GSOMD$。
3. 计算 $H(H(E_B(PI)) \parallel GSOMD)$ 和 $E_C(DS)$。如果它们相等，银行就确认了持卡者的签名。
4. 计算 $D_B(E_B(PI))$，获得支付说明 PI。
5. 返回一个加密的（用 E_M）数字签名授权给销售商，以保证支付。

销售商按以下步骤完成该程序：

1. 回复一个加密的数字签名（使用 E_C）收据给持卡者，表明交易已经完成。

销售商仅能看到付款命令的加密形式 $E_B(PI)$，看不到信用卡号码。由于是用散列函数来计算 DS，所以销售商或银行想要更改有关该命令的任何信息都是不可行的。

银行只能看到商品和服务要求的消息摘要，并不知道定购的内容。

这个过程要满足完整性、私有性和安全性的要求。在实际的实施中，为了确保真实性要求更多的步骤，例如，它必须保证所用的公钥真正属于所声明的参与者，而非冒名顶替者，为了这一目的需要一个来自可信权威的证书。

9.2 数字现金

假设国会议员 Bill Passer 获得了来自他朋友 Phil Pockets 的一大笔馈赠，由于很明显的原因，他想隐藏真相，假装这笔钱是来自其他人如 Vera Coode，或者 Phil 并不想让 Bill 知道这笔钱的来源，如果 Phil 用支票来支付，银行的良好记录将会暴露他，同样，议员 Passer 不能通过信用卡接收支付，只有数字现金可以完成匿名支付。

现在假设 Passer 留在办公室里要处理很多事项，并且我们已经接近 21 世纪的末期。所有的商务都电子化，那样有可能使用数字现金吗？有几个问题出现了，例如，大约在 21 世纪初，复制钞票是可能的，尽管一个细心的消费者可以辨别出复制版与原版的区别。但是，电子信息的拷贝和原版是没有区别的，因此，如果有人利用有效的数字现金可以制作多份拷贝，就需要有方法来防止双重消费。一种方法是中心银行记录下每一个货币，以及谁拥有每个货币。但是如果货币在消费时都记录下来，匿名性就受到了威胁。偶尔地，与银行的通信可能暂时会失败，所以接收到货币的用户能够在不与银行进行联系的情况下证明该账户的合法性也是需要的。

T. Okamoto 和 K. Ohta［Okamoto - Ohta］列举了一个数字现金系统应具备的 6 个特征：

1. 现金能够在计算机网络中安全地传输。
2. 现金不可以被拷贝或重复使用。
3. 现金的使用者可以匿名。如果现金的使用是合法的，无论是接受者还是银行都无权监视支付者。
4. 传输可以离线操作，意味着传输时不需要与中心银行通信。
5. 现金可以转到他人账户。
6. 一笔现金可以分成若干份。

Okamoto 和 Ohta 给出了一个可以满足所有条件的一个体制。David Chaum 也设计出可以满足其中几个条件的体制。下文中，我们将描述一下由 S. Brands 设计的一个可以满足条件 1 ~4的系统［Brands］。

读者一定注意到了这种体制比古老的真币体制要复杂得多，这是因为，正如我们前面所提到的，电子文件基本上不需要任何代价，这点与相对比较难伪造的物理现金恰恰相反。因此，就需要采取一些措施去截获伪造的数字现金，但这意味着数字现金是必须带有用户签名的，那么又如何匿名呢？解决的方法是使用“受限盲签名”，这一过程使得设计更加复杂。

参与者

参与者是银行、支付者和销售商。

初始化

初始化仅执行一次，而且所有的操作都是通过某个权威中心执行的。选择一个大的素数 p，满足 $q=(p-1)/2$ 也是素数（参见练习 8.8）。令 g 为一个模 p 本原根的平方，这就是说 $g^{k_1}\equiv g^{k_2}\pmod p \Leftrightarrow k_1\equiv k_2\pmod q$，选定两个保密的随机指数 g_1 和 g_2，且定义 g_1 和 g_2 为 g 提升为模 p 的这些指数，然后，这些指数就被丢弃（保存这些数也没有用，而且如果一个黑客发现它们的话，该系统就有危险了）。下面的数：

$$g,\quad g_1,\quad g_2$$

是公开的。同样，我们选定两个公开的散列函数，第一个为 H，一个 5 元组的整数作为输入，并输出一个模 q 整数；第二个为 H_0，一个 4 元组的整数作为输入，并输出一个模 q 整数。

银行

银行选择了它的秘密认证数字 x 并计算

$$h\equiv g^x, h_1\equiv g_1^x, h_2\equiv g_2^x\pmod p),$$

数字 h，h_1 和 h_2 由银行公开和确认。

支付者

支付者选择了一个秘密的认证数字 u 并计算帐号

$$I\equiv g_1^u\pmod p),$$

数字 I 传送给银行，并将 I 和能确定支付者身份的信息（例如，姓名、地址等）存储在一起。但是，支付者不把 u 发送到银行。银行将

$$z'\equiv(Ig_2)^x\pmod p)$$

送回给支付者。

销售商

销售商选择一个标识性的数字 M 并在银行注册它。

创建一个货币

支付者联系银行，申请一个货币。银行要求其身份的证明，就像某人在账户上提取传统性质的现金一样，在当前的体制中所有的货币有相同的价值。每一笔货币都将由 6 元组来表示

$$(A,B,z,a,b,r)。$$

这看起来相当复杂，但是我们将看到这种付出可以保证匿名并同时防止了双重消费。

下面显示了如何来构造这些数字：

1. 银行选择一个随机的数字 w（每个货币对应不同的数字），计算

$$g_w\equiv g^w \text{和} \beta\equiv(Ig_2)^w\pmod p),$$

并将 g_w 和 β 发送给支付者。

2. 支付者选择了一个秘密的随机的 5 元整数组

$$(s,x_1,x_2,\alpha_1,\alpha_2)。$$

3. 支付者计算

$$A\equiv(Ig_2)^s,\ B\equiv g_1^{x_1}g_2^{x_2}, z\equiv z'^s,$$

$$a\equiv g_w^{\alpha_1}g^{\alpha_2},\ b\equiv\beta^{s\alpha_1}A^{\alpha_2}\pmod p)。$$

货币中用 $A=1$ 表示不允许，仅有两种途径会导致其发生，一种是当 $s\equiv0\pmod q$，所以我

们要求 $s\neq0$；另一种是当 $Ig_2\equiv1 \pmod p$，这就意味着支付者已经通过幸运的选择 u 值解决了离散对数问题，这里素数 p 的选择应该大到这种情况不可能发生。

4. 支付者计算出

$$c\equiv\alpha_1^{-1}H(A,B,z,a,b) \pmod q,$$

然后发送 c 给银行。这里 H 是前面提到的公开散列函数。

5. 银行算出 $c_1\equiv cx+w \pmod q$ 并将 c_1 发送给支付者。

6. 支付者算出

$$r\equiv\alpha_1c_1+\alpha_2 \pmod q。$$

这样货币（A，B，z，a，b，r）的建立就完成了。该货币的数量是从支付者的银行账户中扣除得到的。

在每次有支付者想要建立一个货币时，这个过程就会重复执行。银行对每笔交易选择一个新的随机数 w，同样，每一个支付者对每一个货币也选择一个新的 5 元组(s，x_1，x_2，α_1，α_2)。

消费货币

支付者将货币（A，B，z，a，b，r）传送给销售商。以下就是其执行的过程：

1. 销售商检查是否满足

$$g^r\equiv ah^{H(A,B,z,a,b)} \qquad A^r\equiv z^{H(A,B,z,a,b)}b \pmod p。$$

如果满足，销售商认为该货币合法。然而需要有更多的步骤来防止双重消费。

2. 销售商计算

$$d=H_0(A,B,M,t),$$

这里 H_0 是在初始化阶段选定的散列函数，t 是一个表示交易的日期和时间的数值。这里加上参数 t 是因为不同的交易会有不同的 d 值，销售商将 d 发送给支付者。

3. 支付者计算

$$r_1\equiv dus+x_1, r_2\equiv ds+x_2 \pmod q,$$

这里 u 是支付者的密码，s，x_1，x_2 是之前选定的保密随机 5 元组。支付者发送 r_1 和 r_2 给销售商。

4. 销售商检查是否满足

$$g_1^{r_1}g_2^{r_2}\equiv A^dB \pmod p。$$

如果该等式成立，销售商就接受该货币；否则，销售商就拒绝它。

销售商将货币存放在银行

收到货币几天以后，销售商想把它存在银行里。销售商提交货币（A，B，z，a，b，r）加上 3 元组（r_1，r_2，d）的结果。银行按照如下步骤进行：

1. 银行检验货币（A，B，z，a，b，r）之前是否未存储起来。如果没有，紧接着执行下一步骤。如果已经存储，银行将跳到下一小节所讨论的欺骗控制步骤。

2. 银行检验是否满足：

$$g^r\equiv ah^{H(A,B,z,a,b)} \quad A^r\equiv z^{H(A,B,z,a,b)}b \text{ 和 } g_1^{r_1}g_2^{r_2}\equiv A^dB \pmod p,$$

如果满足，货币是有效的，销售商的账户是可信的。

欺骗控制

有几种可能使得一些人试图进行欺骗。下面是对付他们的方法：

1. 支付者双重消费，一次是跟销售商，一次是跟我们所称的卖主。销售商使用 3 元组

(r_1, r_2, d) 提交货币，卖主使用 (r_1', r_2', d') 提交货币。简单的计算说明

$$r_1 - r_1' \equiv us(d - d'),\ r_2 - r_2' \equiv s(d - d') \pmod q。$$

两式相除产生结果 $u \equiv (r_1 - r_1')(r_2 - r_2')^{-1} \pmod q$。银行计算出 $I \equiv g_1^u \pmod p$ 并鉴别支付者。因为银行不可能发现 u，否则，这就证明（至少也会怀疑）双重消费的发生。那么支付者就会被送入监狱（如果陪审团认为离散对数问题是困难的）。

2. 销售商试图将货币提交两次，一次是合法的 3 元组 (r_1, r_2, d)，一次是伪造的 3 元组 (r_1', r_2', d')。这对于销售商来说基本上是不可能的，因为销售商很难产生满足

$$g_1^{r_1'} g_2^{r_2'} \equiv A^{d'} B \pmod p$$

的数。

3. 有人试图制造出没有授权的货币。这要求找到满足 $g^r \equiv ah^{H(A,B,z,a,b)}$ 和 $A^r \equiv z^{H(A,B,z,a,b)} b$ 的数，这是相当困难的。例如，已知 A，B，z，a，b，然后要找出 r，要求解一个离散对数问题，才能解出第一个等式。我们注意到，因为 x 的值只有银行知道，那么支付者就可以企图利用一个新的 5 元组再产生第二个货币，所以找到正确的 r 值是很困难的。

4. 一个叫伊芙 L. 迪尤尔的恶意商人从支付者那里收到一个货币并将其存入银行，他还想和销售商一起花费该货币。伊芙将货币给销售商后，销售商计算 d'，它与 d 很相近但并不相等。伊芙不知道 u，x_1，x_2，s，但她必须选择 r_1'和 r_2'以满足 $g_1^{r_1'} g_2^{r_2'} \equiv A^{d'} B \pmod p$，这又是一个离散对数问题。伊芙为何不可以直接用她已经知道的 r_1，r_2 呢？这是由于 $d' \neq d$，销售商会发现 $g_1^{r_1} g_2^{r_2} \not\equiv A^{d'} B$。

5. 工作在银行的某一员工试图伪造一个货币。该员工所知的信息与伊芙基本相同，只是多了一个身份标识数 I，于是就有可能造出一个货币满足$g^r \equiv ah^{H(A,B,z,a,b)}$，然而，由于支付者将 u 保密，该员工就不能找到一个合适的 r_1。当然，如果允许 $s = 0$，这就有可能了，这也正是不允许 $A = 1$ 的一个原因。

6. 某人从支付者那里盗取了货币并试图消费它，这时第一个认证方程同样满足，只是该盗用者不知道 u 的值，从而不能得到满足 $g_1^{r_1} g_2^{r_2} \equiv A^{d'} B$ 的 r_1 和 r_2。

7. 恶意商人伊芙 L. 迪尤尔在货币和 (r_1, r_2, d) 提交给银行前从销售商那里盗取了它们，除非银行要求销售商记录下每次交易的时间和日期从而可以再现产生 d 的输入，否则伊芙的盗用将会成功。当然这一点也是普通现金交易的一个缺点。

匿名

在与销售商进行交易的整个过程中，支付者从未提供任何身份证明，这与传统的现金交易是相同的。同样，注意到银行在销售商存储前绝不知道其货币的 A，B，z，a，b，r 的值。事实上，银行仅仅提供了两个数 w 和 c_1，并且只知道 c 值。然而在双重消费情况下，货币仍然包含了识别支付者的信息。销售商和银行能否从货币 (A, B, z, a, b, r) 和 3 元组 (r_1, r_2, d) 中获取支付者的身份呢？由于银行还知道支付者的身份标识 I，所以认为银行能在任何情况下试着识别出支付者是可能的。由于只有支付者知道保密随机数 s，x_1，x_2，A 和 B 也是随机数，在特殊情况下，A 是g 的一个随机幂且不能用来推导出 I。数 z 就等于 $A^x \pmod p$，所以它们不能提供任何除了 A 以外的帮助，由于 a 和 b 产生两个新的保密的随机指数 α_1，α_2，它们在除了支付者以外的人看来同样是随机数。

此时，有 5 个数 A，B，z，a，b 在除了支付者以外的人看来如同是 g 的随机次幂，然而，当发送 $c \equiv \alpha_1^{-1} H(A,B,z,a,b) \pmod q$ 给银行后，银行会试着算出 H 值从而推出 α_1。但

是银行并未得到货币，于是无法算出 H 的值，然而银行可以试着记下其所接收到的所有 c 值的一个表单，结合已存入银行的每个货币的 H 值，然后试试所有的组合以找到 α_1。但是很容易发现，在一个包含上千万个货币的系统中，α_1 的可能值的数量太大以致于此方法并不实用。所以在知道 b 之后得到 c 的值也不大可能帮助银行识别支付者。

数 α_1 和 α_2 提供了对货币的所谓**受限盲签名（restricted blind signature）**的印记。也就是说，使用一次货币系统不会考虑签名者（即支付者）的身份，但是使用两次则会考虑（这样一来就如前所述，支付者会被送到监狱里）。

要看受限盲签名的效果，我们可以通过假定 $\alpha_1 = 1$，而从根本上将 α_1 从该过程中去掉。这样银行就能记录下一个 c 值的列表，附带 c 所对应的用户。当存储了一个货币后，将会算出 H 的值并与该列表进行比较。有可能所给定的 c 值仅有一个用户与之对应，这样银行就能识别出支付者。

9.3 习　题

1. 说明一个有效的货币满足下列确认等式：

$$g^r \equiv a h^{H(A,B,z,a,b)}, \quad A^r \equiv z^{H(A,B,z,a,b)} b \quad 和\ g_1^{r_1} g_2^{r_2} \equiv A^d B \pmod p。$$

2. 一个黑客发现了银行的保密数字 x。说明一下货币是如何在没有银行账户的情况下产生和支付的。

3. 数字 g_1 和 g_2 是 g 的幂值，但假设其指数是很难被发现的。假设我们令 $g_1 = g_2$。

（a）证明如果支付者用 r_1'，r_2' 来替换 r_1，r_2，且使得 $r_1 + r_2 = r_1' + r_2'$ 成立，那么确认等式仍然成立。

（b）证明如果支付者没有被鉴别可能会双重消费。

4. 假设货币只用（A，B，a，r）来表示，例如忽略 z 和 r，使用只含 A，B，a 的方程的散列函数 H，并忽略确认等式 $A^r \equiv z^H r$。说明支付者可能会将 u 的值改成任何他想要的数（不通知银行），计算一下 I 的新值，产生一个货币使之符合剩余的两个确认等式。

5. 如果支付者双重消费，一次跟销售商，一次跟自动售货机，为什么很可能是 $r_2 - r_2' \not\equiv 0 \pmod q$（这里 r_2，r_2' 正如在欺骗控制的讨论中所提到的）？

秘密共享方案

第 10 章

如果你乐意，我们假设你在网上炒股获得了几十亿美元而且你想把这些财产留给你的亲戚们。你的钱被锁进了只有你知道的安全的地方，你并不想把秘密告诉你的 7 个孩子中的任何一个，因为他们并不可信。你想把秘密分拆开来，从而让他们中的 3 个拼在一起才能重建真实。那样的话，如果有人想得到遗产就必须和其余两个孩子合作。在这一章中我们将介绍如何解决这类问题。

10.1 秘密分拆

我们列举的第一种情况是最简单的。假想你有一个消息 M，再假设它是一个整数，你想将其拆开给艾丽斯和鲍勃，从而使他们中的任何一个人都不能单独重建消息 M。这个问题的直接解决方法是：给艾丽斯一个随机整数 r，给鲍勃数 $M-r$。为了重建消息 M，艾丽斯和鲍勃只需将他们各自的部分相加即可。

由此产生了一些技术问题，例如所有的整数都是等可能的，所以选择一个随机的整数是不太可能的（对于每个整数，无限多相等概率的和不可能等于 1）。因此，我们选择一个大于所有可能消息 M 的整数 n，并且将 M 和 r 看作是模 n 数。那么选择 r 作为一个随机的模 n 整数就没有问题了，简单地指定每个模 n 整数的概率是 $1/n$。

现在让我们检验一个例子，我们想将一个秘密分拆给 3 个人，艾丽斯、鲍勃和查尔斯。使用前面的思想，我们选择了两个随机的模 n 数 r 和 s，并把 $M-r-s \pmod n$ 给了艾丽斯，把 r 给了鲍勃，把 s 给了查尔斯。为了重建消息 M，艾丽斯、鲍勃和查尔斯只要将其各自的部分相加即可。

对于更一般的例子，如果我们想将秘密 M 拆分给 m 个人，那么我们必须选择 $m-1$ 个随机的模 n 数 $r_1, \dots, r_{m-1}$ 并把他们给 $m-1$ 个人，把 $M-\sum_{k=1}^{m-1} r_k \pmod n$ 给余下的人。

10.2 门限方案

在前一部分中，我们展示了如何将秘密分拆给 m 个人，因此需要所有的 m 个人来重建秘密。在本节中，我们介绍允许由其中一部分人来重建秘密的方法。

曾经有一篇报道说，俄罗斯的核武器控制部门雇佣了一个安全机构，需要其中 2/3 的重要人物才能发射导弹。这个主意并不是很特别，它实际上就是一个通常在间谍片中看到的情节。我们可能会想到一个有 3 个密钥槽的控制面板和需要插人其中的两个密钥且同时打开才能发射导弹来毁灭地球的导弹发射协议。

为什么不只用前一节提到的秘密分拆方案？假设某个国家准备一周之内袭击敌国，秘密被分拆给了 3 个官员。一种秘密分拆方法在需要重构发射秘密时将同时需要他们 3 个人。这种情况有时是不可能的，3 人中的其中一个可能因与前一周的对手言和而进行的外交任务外出，或可能因持不同政见而直接拒绝说出秘密。

定义：令 t，w 为正整数且 $t \leqslant w$。(t,w)-**门限方案**（threshold scheme）是这样一种方法：在 w 个参与者组成的集体中共享消息 M，这样由任何 t 个参与者组成的子集都能重构消息 M，但是小于 t 个参与者组成的子集将无法重构 M。

(t, w) -门限方案是更多普遍的共享方案的关键构成模块，在本章的练习中会研究其中的一些方案。下面我们将讲述建立 (t, w) -门限方案的两种方法。

第一种方法是在 1979 年由 Shamir 发明的，称为 **Shamir 阈值方案**（Shamir threshold scheme）或拉格朗日插值法。这是基于我们在高中代数中一些思想的自然推广，诸如两点决定一条直线，三点决定一个二次方程式等等。

选定一个素数 p，p 应该较所有可能的消息大并且比参与者的个数 w 大。所有的计算都会执行模 p 的操作。该素数代替了 10.1 节中的整数 n。如果这里用一个合数来替代，那么我们得到的矩阵可能是不可逆的。

消息 M 用一个模 p 数来表示，我们想要在 w 个人中进行拆分，按照这种方式要重构该消息将需要其中的 t 个人。我们做的第一件事就是随机地选定 $t-1$ 个模 p 数，称为 s_1、s_2，$\cdots s_{t-1}$。这样得到多项式：

$$s(x) \equiv M + s_1 x + \cdots + s_{t-1} x^{t-1} \pmod p$$

该多项式满足 $s(0) \equiv M \pmod p$。现在，对于 w 个参与者，我们选定不同的整数 $x_1, \ldots, x_w \pmod p$，并给每名参与者一个数对 (x_i, y_i)，其中 $y_i \equiv s(x_i) \pmod p$。例如，合理选择 $1, 2, \ldots, w$ 为 x 的值，这样我们就产生数对 $(1, s(1)), \ldots, (w, s(w))$，给每人一个数对。所有人都知道素数 p 的值，而多项式 $s(x)$ 则是保密的。

现在假设 t 个人聚集并分享各自的数据对，为了简化符号，我们假设数据对为 (x_1, y_1)，$\cdots$，(x_t, y_t)，他们想要重构消息 M。

我们从线性逼近开始。假设我们有一个 $t-1$ 阶的多项式 $s(x)$，我们想通过点 (x_1, y_1)，$\cdots$，(x_t, y_t) 重构 M，其中 $y_k = s(x_k)$。这意味着：

$$y_k \equiv M + s_1 x_k^1 + \cdots + s_{t-1} x_k^{t-1} \pmod p, 1 \leqslant k \leqslant t,$$

如果我们指定 $s_0 = M$，那么我们可以重写上式如下：

$$\begin{pmatrix} 1 & x_1 & \cdots & x_1^{t-1} \\ 1 & x_2 & \cdots & x_2^{t-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_t & \cdots & x_t^{t-1} \end{pmatrix} \begin{pmatrix} s_0 \\ s_1 \\ \vdots \\ s_t \end{pmatrix} \equiv \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_t \end{pmatrix} \pmod p。$$

这是一个范德蒙（Vandermonde）矩阵，我们把它记为 V。我们知道，如果这个方案的矩阵

V 是模 p 非零的，则其有惟一的模 p 解（见 3.8 节）。能够证明它的行列式为：

$$\det V = \prod_{1 \leq j < k \leq t} (x_k - x_j),$$

仅当两个 $x_i \bmod p$ 相等时行列式为 0 mod p（此时我们需要 p 为一个素数，见练习 3.3）。因此，只要我们有不同的 x_k 的值，方案就有惟一解。

现在我们来描述一个二次逼近，其导出一个重构多项式的公式并因此重构秘密。我们的目的是由值 (x_k, y_k) 中已知的 t 重构多项式 $s(x)$ 。首先，令

$$l_k(x) = \prod_{\substack{i=1 \\ i \neq k}}^{t} \frac{x - x_i}{x_k - x_i} \pmod p,$$

这里，我们使用在 3.3 节中描述的模 p 分数。那么

$$l_k(x_j) \equiv \begin{cases} 1 & \text{当 } k = j \text{ 时} \\ 0 & \text{当 } k \neq j \text{ 时。} \end{cases}$$

这是因为 $l_k(x_j)$ 是所有因子 $(x_k - x_i)/(x_k - x_i)$ 的乘积，所有的因子都为 1。当 $k \neq j$ 时，乘积 $l_k(x_j)$ 中包含为零的 $(x_j - x_j)/(x_k - x_j)$ 因子。

拉格朗日插值多项式（Lagrange interpolation polynomial）为：

$$p(x) = \sum_{k=1}^{t} y_k l_k(x)$$

当 $1 \leq k \leq t$ 时，满足 $p(x_j) = y_j$ 。例如：

$$p(x_1) = y_1 l_1(x_1) + y_2 l_2(x_2) + \cdots \equiv y_1 \cdot 1 + y_2 \cdot 0 + \cdots \equiv y_1 \pmod p),$$

我们通过前面的范德蒙矩阵的逼近知道，多项式 $s(x)$ 是惟一一个可以满足指定值的 $t-1$ 阶多项式。因此，$p(x) = s(x)$ 。

现在，为了重构秘密消息，我们所要做的就是计算 $p(x)$ 并在 $x = 0$ 处验证它。下面是给出的公式：

$$M \equiv \sum_{k=1}^{t} y_k \prod_{\substack{j=1 \\ j \neq k}}^{t} \frac{-x_j}{x_k - x_j} \pmod p\text{。}$$

例：让我们创建一个（3，8）-门限方案，所以我们有 8 个人并且希望任意 3 个可以确定秘密，但是两个人是无法确定任何信息的。

假设秘密是数字 $M = 190503180520$（对应单词"secret"）。选择一个素数 p，如 $p = 1234567890133$（我们需要一个比秘密大的素数，但是没有必要选择一个比秘密大很多的素数）。随机选择模 p 数 s_1 和 s_2 并构造多项式

$$s(x) = M + s_1 x + s_2 x^2\text{。}$$

例如，我们计算

$$s(x) = 190503180520 + 482943028839x + 1206749628665x^2\text{。}$$

现在我们给 8 个人数对 $(x, s(x))$。没有必要随机地选择 x 的值，所以我们可以简单地使用 $x = 1, 2, \ldots, 8$。因此，我们分配下列几个数对，每人一对：

(1,645627947891)

(2,1045116192326)

(3,154400023692)

(4,442615222255)

$$(5,675193897882)$$
$$(6,852136050573)$$
$$(7,973441680328)$$
$$(8,1039110787147)。$$

假设第 2，3，7 个人想合作来确定秘密。我们使用拉格朗日插值多项式。他们计算了通过他们三点的多项式，如下：

$$20705602144728/5 - 1986192751427x + (1095476582793/5)x^2$$

此时他们意识到他们应该在对 p 取模的方式下计算。而

$$740740734080 \times 5 \equiv 1 \pmod p,$$

所以他们用 1/5 代替了 740740734080，像 3.3 节中一样。通过 mod p 来得到

$$190503180520 + 482943028839x + 1206749628665x^2,$$

当然，这是一个原始的多项式 $s(x)$ 。他们关心的只有作为秘密的常数项 190503180520（前面的多项式计算的最后一部分是可以稍微缩短的，因为他们只需要常数项，而不是整个多项式）。

同样，任何 3 个人都可以重构多项式并获取秘密。

如果第 2，3，7 个人选择了线性系统逼近，那么他们需要解决以下问题：

$$\begin{pmatrix} 1 & 2 & 4 \\ 1 & 3 & 9 \\ 1 & 7 & 49 \end{pmatrix}\begin{pmatrix} M \\ s_1 \\ s_2 \end{pmatrix} \equiv \begin{pmatrix} 1045116192326 \\ 154400023692 \\ 973441680328 \end{pmatrix} \pmod{1234567890133}。$$

这将得到：

$$(M,s_1,s_2) \equiv (190503180520,482943028839,1206749628665),$$

所以他们可以恢复多项式和消息。

如果只有两个人合作会发生什么？他们可以获取什么信息？举个例子，假设第 4 个人和第 6 个人分别拥有（4，442615222255）和（6，852136050573）。令 c 为任意可能的秘密，$ax^2 + bx + c$ 是惟一一个通过（0，c），（4，442615222255），（6，852136050573）三点的多项式。因此，任何秘密都有可能出现。

同样，他们猜不到其他人所持有的部分，比如，对于第 7 个人，任何一个点$(7,y_7)$产生惟一的秘密 c，而任意秘密 c 产生一个多项式 $ax^2 + bx + c$，对应的就是 $y_7 = 49a + 7b + c$。因此，y_7 可能会出现任何值，而且每一个都对应一个秘密。所以第 4 和第 6 个人在只有他们两个点的情况下不会获得更多关于秘密的信息。

相似地，如果我们使用 $t-1$ 阶的多项式，$t-1$ 个人不可能在只有他们的数据的情况下获得更多关于秘密的信息。因此，需要 t 个人来确定秘密。

还有另外一种方法可以用来共享秘密。我们现在描述 Blakley 提出的方法，同样是 1979 年诞生的。假设有几个人，我们想安排任意 3 个都可以找出秘密，但是任意两个不可以。选择一个素数 p，令 x_0 为秘密。随机选择 y_0，z_0 mod p。因此，我们在三维空间内就有了一个点 $Q=(x_0,y_0,z_0)$。每个人都给了一个通过 Q 点的平面方程，是按下列方法完成的。随机选择了模 p 数 a，b，那么令 $c \equiv z_0 - ax_0 - by_0 \pmod p$，则平面为

$$z = ax + by + c,$$

每一个人都如此。通常，3 个平面汇于一点，其必为 Q。两个平面交于一条直线，所以通常

没有关于秘密 x_0 的信息可以获取（见练习 11）。

注意，只有一个坐标可以用来承载秘密。如果这个秘密是在所有的 3 个坐标 x_0，y_0，z_0 中发布的，那么只有对应于两个人平面的交线上一个点的信息是有意义的。

3 个人来推导秘密的方法如下，共有 3 个等式：

$$a_i x + b_i y - z \equiv -c_i \pmod p, 1 \leqslant i \leqslant 3,$$

这将产生矩阵等式：

$$\begin{pmatrix} a_1 & b_1 & -1 \\ a_2 & b_2 & -1 \\ a_3 & b_3 & -1 \end{pmatrix} \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} \equiv \begin{pmatrix} -c_1 \\ -c_2 \\ -c_3 \end{pmatrix},$$

只要矩阵的行列式模 p 非零，矩阵就是模 p 可逆的，而如果矩阵模 p 可逆的，那么就可找到秘密 x_0（当然，实际上可以通过解决行变换来获取秘密，而不是逆置这个矩阵）。

例：令 $p=73$，假设我们给出 A，B，C，D，E 的以下几个平面：

$$A: z = 4x + 19y + 68$$
$$B: z = 52x + 27y + 10$$
$$C: z = 36x + 65y + 18$$
$$D: z = 57x + 12y + 16$$
$$E: z = 34x + 19y + 49。$$

如果 A，B，C 想重构秘密，他们计算：

$$\begin{pmatrix} 4 & 19 & -1 \\ 52 & 27 & -1 \\ 36 & 65 & -1 \end{pmatrix} \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} \equiv \begin{pmatrix} -68 \\ -10 \\ -18 \end{pmatrix} \pmod{73},$$

其结果是$(x_0, y_0, z_0) = (42, 29, 57)$，所以秘密是 $x_0 = 42$。同样，A，B，C，D，E 中的任意 3 个可以联合来重构秘密 x_0。 ■

通过在 t 维空间用 $t-1$ 维的超平面，我们可以对任意的 t 和 w 使用同样的方法来创建一个 (t,w)－门限方案。

只要 p 是合理的大，很可能矩阵是可逆的，虽然这并没有保证。选择 a，b，c 使矩阵总是可逆的并不是很困难。所以，在 Shamir 方法中这个条件通常会满足。两种方法的矩阵等式是相似的，并且 Shamir 方法可以看作是 Blakley 方法的特殊情况。但是因为 Shamir 方法服从范德蒙矩阵，所以始终可以解这些等式。另外一种高级的 Shamir 方法要求每个人拥有比较少的信息：(x, y) 相对 $(a, b, c, \ldots)$ 拥有比较少的信息。

现在我们回到 Shamir 方法并考虑基本情况的变动。通过给某些人更多的部分，就可以使某些人比其他人更重要。举个例子，假设我们有一个方案，其中需要 8 个部分来获取秘密，并假设给了老板 4 部分，给了他的女儿们 2 部分，其他职员一人一部分。那么，老板和其中的两个女儿就可以获取秘密，或者三个女儿和两个普通职员也可以获取秘密。

这里有一个更为复杂的情况。假设两个公司 A 和 B 分享一个金库。他们想拥有一个方案，其中需要有来自 A 公司的 4 名和来自 B 公司的 3 名人员来获得秘密的重构。显然，如果我们只简单地提供同一秘密的部分是不行的，因为一个公司可能不需要另一公司而使用它的员工的所有部分就可以获得秘密。下面是其解决方法。记秘密 s 为两数之和 $s \equiv c_A +$

$c_B \pmod p$)。现在将 c_A 作为一个二次多项式的常数项拆分到 A 的雇员中。类似地，令 c_B 是一个二次多项式的常数项，并将其拆分到 B 的雇员中。如果 A 中的 4 个雇员和 B 中的 3 个雇员合作，即来自 A 公司的雇员决定 c_A，来自 B 公司的雇员决定 c_B。将 c_A 和 c_B 相加即得出 s。

注意，A 通过其自身没有获取关于秘密 s 的任何信息。因为 $c_A + x \equiv s \pmod p$，对每一个 s 只有惟一的解 x，所以 s 的每一个可能的值都对应于一个可能的 c_B 值。因此，知道了 c_A 对找到秘密没有帮助，A 同样需要知道 c_B。

10.3 习　题

1. 假设你有一个值为 5 的秘密。你想创建一个方案，将秘密拆分给 A，B，C，D 四个人，从而使他们中的任何两个人都可以确定秘密，但是任何一个人单独是无法确定的。描述一下这是如何做到的。请特别列举你给每个人来完成秘密的信息（即数字）。

2. 你创建了一个（2，30）的 Shamir 阈值方案，并将工作在模素数 101 下。其中两部分是（1，13）和（3，12）。另外一个人得到了（2，*），但是 * 是未知的。* 的真实值是多少？

3. 在一个(3,5)的 Shamir 秘密拆分方案中，用模数 $p=17$ 将(1,8)，(3,10)，(5,11) 分发给艾丽斯、鲍勃和查尔斯。计算一下相应的拉格朗日插值多项式，并确定秘密。

4. 在一个 Shamir 秘密拆分方案中，秘密是一个四次多项式的常数项模素数 1093 的余数。假设 3 个人有秘密（2,197)，(4,874)，(13,547)。有多少种可能的秘密值？

5. 马克不喜欢取模运算，所以他想在没有它们的情况下执行一个（2,30)Shamir 秘密共享方案。他的秘密是 M(一个正整数)，并且他给了人 i 第$(i, M+si)$ 部分，其中 s 是他随机选择的一个正整数。鲍勃接收了(20,97）这一部分。描述鲍勃如何确定出 M 的可能性，并确定 M 可能是哪些值？

6. 一个密钥发布者使用（2，20）－门限方案给 20 个参与者发布了一个电子保险箱联合体。

(a) 如果一个未知参与者是骗子，他会出示一个随机部分，那么打开保险箱所需参与者的最少人数是多少？

(b) 如果他们只允许试验一次联合体（如果他们错了，保险箱就会永远关闭)，那么至少需要多少个参与者来打开保险箱？(注意：这似乎有一点微妙。多数票事实上是由 4 个人决定的，但是你需要说明不会出现平局。)

7. 某军事办公室由一个将军、两个上校和五个办事员组成。他们享有导弹的控制权，但是只有将军想发射或者是五个办事员决定发射，或是两个上校决定发射，或者是一个上校和三个办事员决定发射时才可以发射。描述一下你会怎样处理此秘密拆分方案。(提示：试用（10，30）Shamir 方案来发布。)

8. 假设有 4 个人在一间房间里，他们中有一个是外国的密探。另外 3 个都给了对应于 Shamir 秘密共享方案的秘密部分，且任意两个人都可以确定此秘密。这个外国密探随机地选择了一对，其中人员与秘密的对应如下。这些数都是工作在对 11 取模的情况下。

$$A:(1,4)\quad B:(3,7)\quad C:(5,1)\quad D:(7,2)$$

确定哪一个是外国密探，并说明秘密消息是什么。

9. 考虑如下的情况：A政府、B政府和C政府是互相敌对的，但是又共同受到来自南极洲迫近的危险。他们都派出了由10名成员组成的代表团参加国际高级会议，来磋商南极洲企鹅造成的世界性安全威胁问题。他们决定对自己的对手采取一种观察的态度。但是，他们又决定如果这些鸟实在是太吵的话，那么就对南极洲采取全面的出击。使用秘密共享方案技术，描述一下他们怎样分配运行秘密，使得A代表团的3个人、B代表团的4个人以及C代表团的2个人合作就可以重建运行秘密。

10. 这个问题探讨了所说的牛顿插值形式。在Shamir方法中，我们列举了两种计算插值多项式的方法。这种等式逼近方案解决困难但是计算简单，而用拉格朗日逼近来确定插值多项式很简单，但是计算复杂。插值多项式的牛顿形式是选择1，$x-x_1$，$(x-x_1)(x-x_2)$，…，$(x-x_1)(x-x_2)\cdots(x-x_t)$ 为基础。插值多项式是：$p(x)=c_0+c_1(x-x_1)+c_2(x-x_1)(x-x_2)+\cdots+c_t(x-x_1)(x-x_2)\cdots(x-x_t)$。演示一下我们可以通过方案 $N_c=y$ 解决冲突 c_k。矩阵 N 有什么专有的属性？为什么这个矩阵使方案易于解决？

11. 在Blakley（3，w）方案中，假设分别给了A和B两人两个平面 $z=2x+3y+13$，$z=5x+3y+1$。证明他们可以在没有第三个人的情况下重构秘密。

10.4 上机题

1. 艾丽斯、鲍勃和查尔斯各自收到一个使用10.1节中描述的秘密拆分方案的秘密所分离出的部分。假设 $n=2110763$，给艾丽斯的部分是 $M-r-s=1008369$，给鲍勃的部分是 $r=593647$，给查尔斯的则是 $s=631870$。求秘密 M。

2. 对于Shamir（4，7）秘密拆分方案，令 $p=8737$，并令各部分是

$$(1,214),(2,7543),(3,6912),(4,8223),(5,3904),(6,3857),(7,510)。$$

使用其中的4个部分来确定秘密，然后再使用另外的4个部分来检验得到的秘密是否一样。

3. 艾丽斯、鲍勃、查尔斯和桃乐茜使用一个（2，4）Shamir秘密拆分方案，并且使用素数 $p=984583$。假设艾丽斯得到的部分是（38，358910），鲍勃得到的部分是（3876，9612），查尔斯得到的部分是（23112，28774），桃乐茜得到的部分是（432，178067），其中有一部分是错误的。确定哪个是错误的，并求出秘密。

博　弈

第 11 章

11.1 电话掷币

艾丽斯住在安克雷奇，鲍勃住在巴尔的摩。有一个朋友不知道他们已经不在一起了，在他的遗嘱中留给他们一辆汽车。他们如何决定谁将得到车呢？鲍勃给艾丽斯打电话，说他将抛一枚硬币。艾丽斯选择背面，但鲍勃却说："抱歉，是正面。"所以鲍勃得到了车。

出于某种原因，艾丽斯怀疑鲍勃也许做了假（而事实上，鲍勃说的是真话，艾丽斯一说出背面，鲍勃就抛出了他特制两面头像的硬币，所以他不可能说谎）。她（艾丽斯）决定下次再发生这种情况时，她将用一种不同的方法。因此，她去了当地的加密专家那里，加密专家告诉了她下面的方法。

艾丽斯选了两个很大的随机的素数 p 和 q，两个数同余于 3 模 4，她将 p 和 q 保密，而将 $n=pq$ 的结果寄给鲍勃，然后鲍勃取了一个随机的整数 x 并计算 $y\equiv x^2 \pmod n$，他将 x 保密而把 y 寄给艾丽斯。艾丽斯知道 y 有一个模 n 平方根（如果不是这样，她的计算就会发现这个事实，这种情况下，她将起诉鲍勃的欺骗行为），所以她用所了解到的 p 和 q 的情况去找 $y \pmod n$ 的 4 个平方根 $\pm a$，$\pm b$（见 3.9 节）。其中一个是 x，但她不知道是哪一个。她随机选了一个（这就是"抛"），比如说是 b，并将它寄给鲍勃。如果 $b\equiv \pm x \pmod n$，鲍勃将告诉艾丽斯她赢了。如果 $b\not\equiv \pm x \pmod n$，鲍勃就赢了。

艾丽斯		鲍勃
$n=pq$	→	n
y	←	$y\equiv x^2$
$a^2\equiv b^2\equiv y$	→	b
艾丽斯赢	←	$b\equiv \pm x$
或者		或者
鲍勃赢	←	$b\not\equiv \pm x$

但是，艾丽斯问，我怎样才能确定鲍勃没有作弊？如果艾丽斯将 b 寄给鲍勃，并且 $x\equiv \pm a \pmod n$，那么鲍勃就知道了 $y \pmod n$ 的 4 个平方根，所以他就能将 n 因数分解。特别地，$\gcd(x-b, n)$ 给出了 n 的一个非平凡的因子。因此，如果通过计算将 n 因数分解是不可行的话，鲍勃得出因子 p 和 q 的惟一方法就是他的 x 值不是艾丽斯发送的值的正值或负值。

当艾丽斯将 n 寄给鲍勃时，鲍勃并没有更多的信息。所以在这种情况下，他不可能得出 p 和 q。因此，艾丽斯可以通过问鲍勃关于 n 的因数分解来检查鲍勃有没有作弊。

如果艾丽斯企图通过寄给鲍勃一个随机的数，而不是 y 的一个平方根来作弊，将会怎样？这必定会防碍鲍勃将 n 因数分解。鲍勃可通过检查艾丽斯所寄数的平方与 y 同余来阻止这种情况的发生。

假设艾丽斯企图给鲍勃寄 3 个素数乘积的结果来欺骗鲍勃。当然，在游戏的最后鲍勃可以问艾丽斯关于 n 的因数分解的情况。如果艾丽斯出示了两个因子，它们将很快被检查出原始值。但是鲍勃不用担心这种可能性的发生。当 n 是 3 个不同的素数相乘的结果，y 将有 8 个平方根。因此，艾丽斯寄的数有 4 种选择。3 种错误选择的每一种都将允许鲍勃找到 n 的一个非平凡因子。这样艾丽斯赢的可能性将减小到 1/4，所以她不会企图这么做的。

这个过程有一个缺陷。设想鲍勃决定他想输，他可以声称他的 x 的值就是艾丽斯寄给他的值。既然她所知道的只有鲍勃的平方数，该平方数与她的平方数同余，所以艾丽斯对此不会有异议。还有别的过程可以阻止鲍勃故意输的企图，但我们在此不作讨论。

最后，我们要提及一点，找与 3 模 4 同余的素数 p 和 q 并不困难。与 1 模 4 同余的素数的密度和与 3 模 4 同余的素数的密度是相同的。因此，找一个随机的素数 p，如果不是 3 模 4，试试另一个，这个处理过程将很快成功。我们可以类似地找到 q。

例：艾丽斯选了

$$p = 2038074743,\ q = 1190494759。$$

她将

$$n = pq = 2426317299991771937$$

寄给鲍勃，鲍勃取

$$x = 1414213562373095048$$

（这并不像它看起来那样随机，但鲍勃认为平方根的十进制部分看起来是随机的），并计算

$$y \equiv x^2 \equiv 363278601055491705 \pmod n,$$

他将 y 寄给艾丽斯。

艾丽斯计算：

$$y^{(p+1)/4} \equiv 1701899961 \pmod p \text{ 和 } y^{(q+1)/4} \equiv 325656728 \pmod q,$$

所以，她知道

$$x \equiv \pm 1701899961 \pmod p \text{ 和 } x \equiv \pm 325656728 \pmod q。$$

中国剩余定理用 4 种方式将这些放在一起，产生

$$x \equiv \pm 1012103737618676889 \text{ 或 } \pm 937850352623334103 \pmod n。$$

假设艾丽斯将 1012103737618676889 寄给鲍勃，这就是 $-x \pmod n$，所以鲍勃宣布艾丽斯赢了。

假设艾丽斯将 937850352623334103 $\pmod n$ 寄给鲍勃，鲍勃将宣布取胜。通过计算

$$\gcd(1414213562373095048 - 937850352623334103, n) = 1190494759,$$

他将证实他赢了。 ■

11.2 电话扑克

艾丽斯和鲍勃很快就厌倦了通过电话抛硬币，于是决定试试扑克游戏。鲍勃取出一副扑克，洗牌，并将牌分成两份，一份给艾丽斯，一份给自己。现在，他将做什么呢？艾丽斯不让鲍勃看她的牌。同样，她建议他也不要玩整副牌。接着争论出现了，但随后有人建议他们各自选他们自己的牌。赌博是如此地快和激烈。赌过几百个硬币之后（他们根据抛硬币的协议将没用过的硬币保存下来），他们都非常激动，艾丽斯和鲍勃发现他们各自都有一个同花大顺。双方都说对方一定作弊了。幸运的是，他们都中意的密码专家可以帮忙。

这儿就有她的建议，建议不涉及数学术语。鲍勃拿出52个完全一样的盒子，每个盒子里放一张牌，并把每个盒子都锁上。他把盒子倒入一个袋子中并把袋子寄给艾丽斯，艾丽斯选5个盒子，用她的锁锁上，然后将盒子寄回给鲍勃。鲍勃用他自己的锁打开这5个盒子，并把它们给艾丽斯寄回。艾丽斯将她自己的锁打开，找到她的5张牌。然后她再选5个盒子，并把它们寄回给鲍勃。鲍勃用他的锁打开，得到他的5张牌。现在设想艾丽斯想换3张牌，她把3张牌放在一个已弃用的盒子中，用她的锁锁上，再把盒子寄给鲍勃，然后她从剩下的42个盒子中选3个盒子并锁上她的锁，寄给鲍勃，鲍勃打开他的锁并把盒子寄回给艾丽斯，艾丽斯打开她的锁便得到牌。如果鲍勃想换2张牌，他将这两张牌放在另外一个弃用的盒子中锁上，然后把盒子寄给艾丽斯。她选2个牌盒寄给鲍勃，鲍勃打开自己的锁得到牌。然后他们比较每手牌看谁会赢，我们可以假定，艾丽斯会赢。

当这手牌玩过之后，鲍勃想确定艾丽斯没有用8张在玩，所以他想检查艾丽斯把3张牌放在弃用的盒子里的情况。他用锁把盒子锁上，把盒子寄给艾丽斯，让艾丽斯把她的锁打开。由于鲍勃的锁仍在盒子上，所以艾丽斯不能换盒子里的牌。她把盒子寄给鲍勃，鲍勃打开锁便得到艾丽斯弃用的牌（这与标准的扑克游戏不同，鲍勃将看到确实被弃用的牌；在标准的扑克游戏中，鲍勃只看到艾丽斯弃用了3张牌，而不用事后看）。类似地，艾丽斯也可以查看鲍勃弃用的那两张牌。

鲍勃可以让艾丽斯将她的牌寄给他，从而检查艾丽斯是否是用发过的牌在玩这手牌。因为剩余的牌上都有鲍勃的锁（牌在艾丽斯那儿，鲍勃打不开），所以艾丽斯没法换牌。

当然，如果艾丽斯或鲍勃不公正地起诉对方，就会产生很多问题。但忽略这些，艾丽斯和鲍勃就可以玩扑克了。但是，将52个盒子寄来寄去的邮资将会削减艾丽斯的利润，所以她去找加密专家请教数学的方法。下面就介绍这种方法。

艾丽斯和鲍勃都同意用一个大素数 p，艾丽斯悄悄地选一个整数 α，使 $\gcd(\alpha, p-1)=1$，鲍勃悄悄地选一个整数 β，使 $\gcd(\beta, p-1)=1$。艾丽斯计算 α'，使 $\alpha\alpha' \equiv 1 \pmod{p-1}$。鲍勃计算 β'，使 $\beta\beta' \equiv 1 \pmod{p-1}$，每一手牌用不同的 α 和 β。每次也可以用不同的 p。

注意到 $c^{\alpha\alpha'} \equiv c \pmod p$，类似地对 β 也是如此。下面将会看到这个性质：$\alpha\alpha' \equiv 1 \pmod{p-1}$，所以对某个整数 k，有 $\alpha\alpha' = 1 + (p-1)k$。因此，当 $c \not\equiv 0 \pmod p$ 时

$$c^{\alpha\alpha'} \equiv c \cdot (c^{p-1})^k \equiv c \cdot 1^k \equiv c \pmod p。$$

一般地，我们在 $c \equiv 0 \pmod p$ 时同样可以有 $c^{\alpha\alpha'} \equiv c \pmod p$。

通过事先确定的方案，我们把52张牌换成52个不同的数 $c_1, \ldots, c_{52} \bmod p$。鲍勃计算

$b_i \equiv c_i^{\beta} \pmod p$，其中 $1 \leqslant i \leqslant 52$，随机地改变这些数的序列，然后寄给艾丽斯，艾丽斯选 5 个数 b_{i_1}，…，b_{i_5}，计算 $b_{i_j}^{\alpha} \pmod p$，其中 $1 \leqslant j \leqslant 5$，并把这些数寄给鲍勃。鲍勃通过提升这些数到幂 β' 来解除他的锁，然后将结果发送给艾丽斯，艾丽斯通过提升到幂 α' 来解除她的锁，由此艾丽斯给出了自己的牌。

接着艾丽斯选择超过 5 个个数 b_i 发回给鲍勃，鲍勃通过提升这些数到幂 β' 来解除他的锁，由此鲍勃给出了自己的牌。这个游戏剩下的过程按这种方式进行。

艾丽斯要推测鲍勃的牌似乎很难。她可以猜测哪个加过密的牌 b_i 和一个确定的未加密的牌 c_j 相对应。这意味着艾丽斯要解形如 $c_j^{\beta} \equiv b_i \pmod p$ 的方程从而得到 b，为了 b 而对 52 种选择都这样做，最多可以给出关于 β 的 52 种选择。然后通过选择另一张牌 c_j' 决定正确的指数 β。我们可以尝试 β 的多种可能性，在加过密的牌的清单上查看哪些给出了加密值。但是艾丽斯要解决的这些方程是离散对数问题，当 p 很大的时候，一般认为这是很难的（见第 7 章）。

例：让我们考虑一个只有 5 张牌的简化的游戏：10，J，Q，K，A。每个人一张牌，拿到最高牌的人就是赢家。把牌换成数字，令 $a=01$，$b=02$，...，所以我们得到下面的情形。

10	J	Q	K	A
200514	10010311	1721050514	11091407	10305

取素数 $p=2396271991$。艾丽斯悄悄地选好她的 $\alpha=1234567$，鲍勃悄悄地选好他的 $\beta=7654321$。艾丽斯计算 $\alpha'=402406273$，鲍勃计算 $\beta'=200508901$。这可以通过扩展的欧几里得运算法则做到。为了确认一下，艾丽斯核对 $\alpha\alpha' \equiv 1 \pmod{p-1}$，鲍勃也会用 β 和 β' 做类似的计算。

现在鲍勃计算（模 p 同余）

$$200514^{\beta} \equiv 914012224$$
$$10010311^{\beta} \equiv 1507298770$$
$$1721050514^{\beta} \equiv 74390103$$
$$11091407^{\beta} \equiv 2337996540$$
$$10305^{\beta} \equiv 1112225809。$$

他将这些牌的顺序打乱，再把它们寄给艾丽斯：

1507298770，1112225809，2337996540，914012224，73490103。

由于艾丽斯并不知道 β，她不可能不通过大量的计算就推算出哪张牌是哪个数。

现在艾丽斯通过从这些数中选一个来选她的牌。例如，第 4 张，求出其 α 次幂，并寄给鲍勃：

$$914012224^{\alpha} \equiv 1230896099 \pmod p,$$

鲍勃通过求出其 β' 次幂来打开他的锁，把它寄回给艾丽斯：

$$1230896099^{\beta'} \equiv 1700536007 \pmod p,$$

现在艾丽斯通过求出其 α' 次幂来解开她的锁：

$$1700536007^{\alpha'} \equiv 200514 \pmod p$$

所以她的牌是 10。

现在艾丽斯通过简单地从她收到的原牌中选一个来选鲍勃的牌。例如，1507298770，并把它寄回给鲍勃。

鲍勃计算：

$$1507298770^{\beta'} \equiv 10010311 \pmod p,$$

所以他的牌是 J。

这就实现了对纸牌的预期处理。现在艾丽斯和鲍勃比较纸牌，鲍勃赢了。为了防止作弊，艾丽斯和鲍勃将他们的秘密指数 α 和 β 公布。设想艾丽斯试图说她有 K，鲍勃很快就能计算 α'，并证实她发给艾丽斯的牌是 10。 ■

怎样作弊

没有一种扑克游戏丝毫不存在作弊的可能性，这儿就说一说如何作弊。

鲍勃去他当地的数论家那儿，数论家告诉他二次剩余的问题。如果同余方程 $x^2 \equiv r \pmod p$ 有一个解，那么数 $r \pmod p$ 就叫作模 p 的**二次剩余（quadratic residue）**，换句话说，r 是一个模 p 的平方数。一个二次非剩余 n 是这样一个整数，满足 $x^2 \equiv n \pmod p$ 无解。

有一种简单的方法来判断某一个数 $z \not\equiv 0 \pmod p$ 是否是二次剩余或非剩余。

$$z^{(p-1)/2} \equiv \begin{cases} +1 \pmod p & \text{如果 } z \text{ 是二次剩余} \\ -1 \pmod p & \text{如果 } z \text{ 是二次非剩余} \end{cases}$$

（见练习 1）。也可由 Legendre 或 Jacobi 符号加上二次相关性来进行确定。

回想一下，我们需要 $\gcd(\alpha, p-1) = 1$ 和 $\gcd(\beta, p-1) = 1$。所以，α 和 β 是奇数。牌 c 被加密为 c^β，并且

$$(c^\beta)^{(p-1)/2} \equiv (c^{(p-1)/2})^\beta \equiv c^{(p-1)/2} \pmod p,$$

由于 $(\pm 1)^{\text{奇数}} \equiv \pm 1$（同余式两边的符号作同样的选择），所以，当且仅当 c^β 是一个二次剩余时，c 是一个模 p 的二次剩余。对应的叙述也可应用于这张牌的 α 和 $\alpha\beta$ 次幂时。

当艾丽斯给鲍勃寄了 5 张牌，这 5 张牌将构成她的牌，鲍勃很快就可以查看这些牌，看看哪些是二次剩余，哪些不是。这就意味着有两个集合 R 和 N。对于艾丽斯的每张牌来说，鲍勃知道牌是在 R 中还是在 N 中，这给了他微弱的优势。例如，假设他需要知道她是否有 Q，并且他断定牌 Q 在 N 中。如果她只有一张牌 N，那么她有这张牌的可能性就很低了。在这种方式下，鲍勃就获得了微弱的优势并开始赢了。

艾丽斯很快去询问她当地的密码分析家，很幸运，当地的密码分析家也知道二次剩余问题。现在，当艾丽斯选择鲍勃的牌时，例如，她可以把他的牌全部安排在 R 中。然后，她就知道他的牌是从 26 张牌中选的，而不是从 52 张牌中选的。这就比鲍勃所知道的部分信息更好，并且这也有足够的用处，相对鲍勃，她可以获得优势。最后，艾丽斯变得非常大胆，艾丽斯偷偷地选了素数 p，以致于黑桃 A，K，Q，J 和 10 是仅有的二次剩余。当她选鲍勃的牌时，她给他 5 张满足非剩余的牌。她给自己选了 5 张满足剩余的牌。鲍勃对每一张牌都要计算剩余和非剩余，由于好几次他的牌要么都有剩余，要么都没有，所以他已经开始怀疑了。但现在他在艾丽斯出牌之前就知道她为自己选择的是一个同花大顺。他指控她作弊，接下来就是协商，最后他们又开始抛硬币。

例：让我们回顾一下简单的例子（只有 5 张牌）。素数 p 的选择不是随机的，事实上，

$$200514^{(p-1)/2} \equiv 1$$
$$10010311^{(p-1)/2} \equiv 1$$
$$1721050514^{(p-1)/2} \equiv 1$$
$$11091407^{(p-1)/2} \equiv 1$$
$$10305^{(p-1)/2} \equiv 1,$$

所以只有 A 是非剩余，而剩余的所有牌都是满足二次剩余的。当艾丽斯选她的牌时，她计算：

$$1507298770^{(p-1)/2} \equiv 1$$
$$1112225809^{(p-1)/2} \equiv -1$$
$$2337996540^{(p-1)/2} \equiv 1$$
$$914012224^{(p-1)/2} \equiv 1$$
$$74390103^{(p-1)/2} \equiv 1。$$

这就告诉她 A 是 1112225809，她求出其 α'次幂，然后把它寄给鲍勃。鲍勃求出其 β' 次幂，然后寄回给艾丽斯。当然，她将发现她的牌是 A。 ■

要得到更多的关于通过电话玩扑克游戏的情况，请看［Fortune-Merritt］。

11.3 习 题

1. 设 g 是对素数 p 的本原根，这意味着数 $1, g, g^2, g^3, \ldots, g^{p-1} \pmod p$ 将产生所有非 0 模 p 的同余类。

(a) 设 i 是一个确定的数，并假设 $x^2 \equiv g^i \pmod p$ 有一个解 x，证明 i 一定是一个偶数。(提示：对于某个 j，记 $x \equiv g^j$。现在利用这样一个事实：当且仅当 $k \equiv l \pmod{p-1}$ 时，$g^k \equiv g^l \pmod p$。) 这就证明了非 0 的平方数对 p 取模恰好等于 $1, g^2, g^4, g^6, \ldots \pmod p$，因此 $g, g^3, g^5, \ldots$ 是模 p 的二次非剩余。

(b) 利用本原根的定义，证明 $g^{(p-1)/2} \not\equiv 1 \pmod p$。

(c) 利用练习 3.4 来证明 $g^{(p-1)/2} \equiv -1 \pmod p$。

(d) 令 $x \not\equiv 0 \pmod p$，如果 x 是一个模 p 的二次剩余，证明 $x^{(p-1)/2} \equiv 1 \pmod p$，如果 x 是一个模 p 的二次非剩余，证明 $x^{(p-1)/2} \equiv -1 \pmod p$。

2. 在带有 $n = pq$ 的硬币抛掷协议中，假设鲍勃寄了一个数 y，不管是 y 还是 $-y$，都没有一个模 n 平方根。

(a) 证明 y 不可能是同一个平方数既模 p 又模 q 的余数。类似地，$-y$ 也不可能是同一个平方数模这两个素数的余数。

(b) 假设 y 不是一个平方数模 q 的余数，证明 $-y$ 是这个平方数模 q 的余数。

(c) 证明 y 是一个平方数对两个素数之一取模的余数，$-y$ 是该平方数对另一素数取模的余数。

(d) 善良的艾丽斯决定纠正鲍勃的"错误"。设想 y 是一个平方数模 p 的余数，$-y$ 是一个平方数模 q 的余数。艾丽斯计算一个数 b，使得 $b^2 \equiv y \pmod p$，且 $b^2 \equiv -y \pmod q$，并把 b 寄给鲍勃（b 有两对选择）。展示一下鲍勃怎样利用这个信息来将 n 因数分解，从而宣布获胜。

3. (a) 设 p 是一个奇素数，证明：如果 $x \equiv -x \pmod p$，那么 $x \equiv 0 \pmod p$。

(b) 设 p 是一个奇素数，假设 $x, y \not\equiv 0 \pmod p$ 且 $x^2 \equiv y^2 \pmod{p^2}$。证明：$x \equiv \pm y \pmod{p^2}$（提示：看看 6.3 节中关于基本定律的证明）。

(c) 假设抛硬币的时候艾丽斯通过选择 $p = q$ 来作弊。证明鲍勃经常输是因为艾丽斯经常返回 $\pm x$。所以在游戏结束时，对鲍勃来说，询问两个素数是很明智的。

零知识证明

第12章

12.1 基本构成

据报道，几年前，一些骗子在一个大型商场安装了一台假的自动取款机。当人们插入银行卡并输入密码时，这台机器就记录下相关的信息，然后反馈出“此卡无效”的信息。之后，这些骗子制作一张伪造的银行卡，用刚才获得的密码，到合法的自动取款机上提取现金。

怎样才能避免发生这种情况呢？在很多情况下，人们为了完成某项交易，不得不暴露他们私人的身份证号和密码。任何人获得了这些密码，结合一些（几乎都是相同的）个人资料（如银行卡上的资料）就可以伪装成这个人。所以就需要有这样一种技术，只需运用一些保密数字而不给盗窃者提供任何可以重新运用的信息。这样，零知识证明便应运而生了。

Quisquater，Guillou 和 Berson［参考 Quisquater 等］等人的一个例子很好地解释了基本的挑战-响应协议。如图 12.1 所示，假设存在一个有一扇门的通道。佩吉（Peggy）（被证明者）要向维克托（Victor）（验证者）证明她能够通过那扇门，而且不必提供给维克托任何有关她是如何穿过门的信息（甚至她是从哪一方穿过的）。过程如下：佩吉进入通道，沿着通道的左边或右边前进。维克托在外边等一会儿，然后进来站在 B 点。他向佩吉发出“左”或“右”的命令。然后佩吉按要求从左侧或右侧来到 B 点。整个协议被重复几次，直到维克托满意为止。当然，在每一回合，佩吉都是随机地选择从哪一边进入，维克托也是随机地选择他要求哪一边。

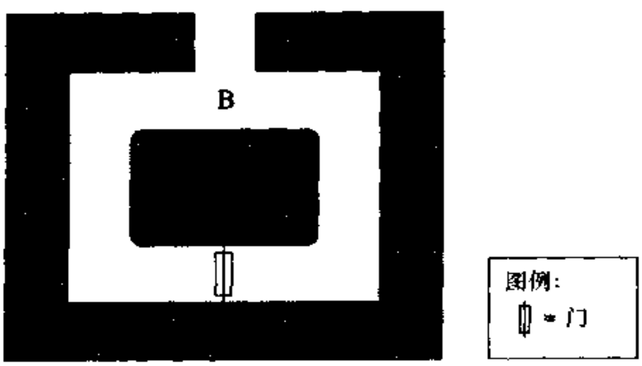


图 12.1　零知识协议中的通道

由于佩吉必须在她获得维克托的命令前选择沿着哪一边走，所以如果她不知道怎样通过门的话，那么她只有50%的机会能够欺骗维克托。因此，如果10次重复中她都走对了的话，那么只有1/1024的可能她不知道如何通过这扇门。在某种意义上，维克托大概就可以相信她，当然，为了确信他也可以多试几次。

假设伊芙正通过安装在B点的一个摄像头来观察整个过程，那么她不可能用看到的任何信息来使维克托或其他人相信她也能通过这扇门。甚至，她都不能证实佩吉能通过这道门。毕竟，佩吉可能已经提前和维克托计划好了左右的序列顺序。由于这个原因，维克托获得的任何有用信息都不可能传送给别人。

注意，在严格的数学意义上，没有任何证据能证明佩吉能通过这道门。但同时也通过一系列的挑战和响应获得了不可抗拒的证据。这就是零知识“证据”的特点。

对于这一过程，有几种数学说法，我们将主要介绍其中之一。令 $n=pq$ 表示两个大素数的乘积，令 y 表示一个平方数模 n 的余数且满足 $\gcd(y,n)=1$，注意求解出模 n 的平方根是比较困难的。实际上，求解模 n 的平方根就相当于对 n 进行因数分解（参考3.9节）。然而佩吉声称知道 y 的平方根 s。维克托想证实它，但佩吉又不想把 s 透露出来。解决方法如下：

1. 佩吉随机选两个数 r_1，r_2，满足

$$r_1r_2 \equiv s \pmod n$$

（当然，她可以随机地选择满足 $\gcd(r_1,n)=1$ 的 r_1，并令 $r_2=s/r_1$，同样，她也可以先选 r_2）。然后计算出 $x_1 \equiv r_1^2, x_2 \equiv r_2^2 \pmod n$，并把 x_1 和 x_2 发送给维克托。

2. 维克托检验 $x_1x_2 \equiv y \pmod n$，然后任意选择 x_1 或 x_2，并让佩吉给出相应的平方根。他核对是否是一个真正的平方根。

3. 重复前两步几次，直到维克托确认为止。

当然，如果佩吉知道 s，整个过程就会很顺利，不会有什么问题。但是，如果佩吉不知道 y 的一个平方根，那么会出现什么情况呢？她仍然可以传送给维克托满足 $x_1x_2\equiv y$ 的两个数字 x_1 和 x_2。如果她知道 x_1 的一个平方根和 x_2 的一个平方根，那么她自然就知道 $y\equiv x_1x_2$ 的一个平方根。因此，x_1 和 x_2 中至少有一个她不知道平方根，那么至少有一半的几率维克托将问到她不知道的那个平方根。由于计算平方根很困难，她不可能提供出需要的答案，因此维克托就会发现她并不知道 s。

然而，假设佩吉猜中了维克托将要问的是 x_2 的平方根。那么她选择一个随机数 r_2，计算 $x_2 \equiv r_2^2 \pmod n$，令 $x_1 \equiv yx_2^{-1} \pmod n$。她传送 x_1 和 x_2 给维克托，所有问题都迎刃而解了。这种方法在任何特定的回合中都给佩吉提供了50%的机会来欺骗维克托，但要求佩吉每次都猜中维克托将问哪个数。只要她猜错了，维克托就会发现她不知道 s。

如果维克托证实佩吉知道一个平方根，那么他能否获得一些可以被其他人使用的信息呢？不能，因为每一步中，他只是获得了一个随机平方数的平方根，而不是 y 的平方根。当然，如果佩吉使用同一个随机数的次数多于一次，他就可能发现 x_1 和 x_2 的平方根，从而知道 y 的平方根。所以佩吉应注意一下随机数的选择。

假设伊芙正在偷听，那么她也只能获得随机数的平方根。如果她想用同样的随机数序列来伪装成佩吉，她便会被问到 x_1 和 x_2的相同序列的精确平方根。例如，如果在某一步里，维克托询问 x_1 的平方根来替代 x_2 的平方根，那么伊芙就提供不出来了。

12.2　Feige-Fiat-Shamir 识别方案

前边的协议要求佩吉和维克托之间相互交流几次，Feige-Fiat-Shamir 法简化了这一过程，采用了一种并行的认证形式。后来这被用作一个识别方案的基础。

同样令 $n = pq$ 表示两个大素数的乘积，佩吉有密码 s_1，…，s_k，令 $v_i \equiv s_i^{-2} \pmod n$（假设 $\gcd(s_i, n) = 1$）。把数字 v_i 传送给维克托，维克托将设法验证佩吉是否知道密码 s_1，…，s_k。佩吉和维克托按如下所示的步骤进行：

1. 佩吉选择一个随机整数 r，计算 $x \equiv r^2 \pmod n$，并把 x 传给维克托。
2. 维克托选择数字 b_1，…，b_k，满足每个 $b_i \in \{0, 1\}$，然后把这些数字传送给佩吉。
3. 佩吉计算出 $y \equiv rs_1^{b_1}s_2^{b_2}\ldots s_k^{b_k} \pmod n$，再把 y 传给维克托。
4. 维克托核对 $x \equiv y^2v_1^{b_1}v_2^{b_2}\ldots v_k^{b_k} \pmod n$。
5. 重复 1～4 步几次（注：每次 r 取不同的值）。

假设 $k = 1$，那么佩吉被要求给出 r 或 rs_1。这是两个随机数，且它们的商是 v_1 的平方根。因此，这与前面讨论的简单方案在本质上是相同的，只不过用商代替了乘积。

下面让我们分析一下 k 取较大值的情况。比如，假设维克托传给佩吉 $b_1 = 1$，$b_2 = 1$，$b_4 = 1$，而其他所有 $b_i = 0$。那么佩吉必须计算出 $y \equiv rs_1s_2s_4$，即 $xv_1v_2v_4$ 的平方根。实际上，在每一个回合中，维克托要求的是形如 $xv_{i_1}v_{i_2}\ldots v_{i_j}$ 的数字的平方根。如果佩吉知道 r，s_{i_1}，…，s_{i_j} 的话，她就能给出相应的平方根。反之，如果她不知道，那么很难计算出平方根。

如果佩吉不知道数字 s_1，…，s_k 中的任何一个（如果另外的人伪装成佩吉也会出现类似的情况），她只能猜测维克托可能发出的数字串。假设佩吉发送 x 之前，她猜对了，那么她就可以选随机数 y，令 $x \equiv y^2v_1^{b_1}v_2^{b_2}\ldots v_k^{b_k} \pmod n$。当维克托把数字串传过来后，佩吉传回 y 的值。当然，验证同余的结果是不出预料的。但是如果佩吉猜错了，那么她必须修改 y 的选择，也就是说她需要计算出 v_i 的一些平方根。

例如：假设当 $b_1 = 1$，$b_2 = 1$，$b_4 = 1$，而其他所有 $b_i = 0$ 时，佩吉能够提供正确的回答，因为这可以通过猜测数字串及用上述方法选择 x 来实现。然而，假设维克托传送出的是 $b_1 = 1$，$b_3 = 1$ 及其他所有 $b_i = 0$，那么佩吉可能准备提供的是 $xv_1v_2v_4$ 的平方根，但被请求提供的却是 xv_1v_3 的平方根。就她所知，这等同于要求她知道 $v_2^{-1}v_3v_4^{-1}$ 的平方根，但这个值她是计算不出来的。在一种极端的情况下，维克托可能传送的所有值都为 0，这意味着要求佩吉必须提供 x 的一个平方根。基于佩吉前面的猜测，这意味着她将知道 $v_1v_2v_4$ 的一个平方根。简而言之，如果佩吉的猜测不正确，她就需要知道 v_i 的非空乘积的平方根，而这她是不能计算出来的。因此，维克托传送的可能数字串有 2^k 个，而其中仅仅有一个允许佩吉欺骗维克托。在这个协议的一个回合中，维克托被骗的可能仅仅为 $1/2^k$。如果这个过程被重复 t 次，那么维克托被骗的几率则为 $1/2^{kt}$，建议取值为 $k = 5$ 和 $t = 4$。这与以前的方案重复 20 次的概率是相同的，所以现行的程序在佩吉和维克托的交流中更有效。当然，维克托并没有获得像佩吉所知道的那么有力的验证信息，比如 s_1 等，但他能够确信伊芙没有冒充佩吉，因为伊芙对 s_i 的值一无所知。

上述可用于建立一个识别方案。令 I 表示一字符串，包括佩吉的姓名、生日和其他一些认为适当的信息。令 H 表示一个公开的散列函数。可信赖的权威人士亚瑟（银行、护照代理…）选择 $n=pq$ 表示两个大素数的乘积。亚瑟计算 $H(I\|j)$，其中 j 取较小的值，$I\|j$ 表示 j 为 I 的补集。通过已知的 p 和 q，他能确定 $H(I\|j)$ 中哪个数有模 n 的平方根，并计算出每个这样的数的一个平方根。这就得到了数 $v_1=H(I\|j_1)$，...，$v_k=H(I\|j_k)$ 及平方根 s_1，...，s_k。数字 I，n，j_1，...，j_k 都公开。亚瑟把 s_1，...，s_k 发送给佩吉，佩吉作为密码保留。一旦平方根被计算出来，素数 p 和 q 就自动作废。同样，一旦 s_1，...，s_k 被传送给佩吉，亚瑟就不必保存它们了。这两点加强了安全性，因为任何侵入亚瑟电脑的人都不能危及佩吉的安全。而且，不同的人使用不同的 n 值，所以一次很难危及几个个体的安全。

注意，由于有一半的数模 p 和模 q 有平方根，中国剩余定理表明有 1/4 的数模 n 有平方根。因此，每一个 $H(I\|j)$ 都有 1/4 的可能性有模 n 的平方根。这意味着亚瑟很快就能计算出必要的数字 j_1，...，j_k。

比如，佩吉来到一个自动取款机前。机器从佩吉的卡上读取 I，从数据库中下载了 n，j_1，...，j_k，并计算 $v_i=H(I\|j_i)$，其中 $1\leqslant i\leqslant k$。然后就执行上述程序来验证佩吉是否知道 s_1，...，s_k。通过几个回合的验证，机器确认该人是佩吉，并允许她提取现金。简单的实现是要求输入一些关于佩吉的信息，但至少伊芙不能得到佩吉的密码。更好的实现是在卡里嵌入芯片，并存储一些信息，用这种方式别人就无法提取这些信息。

如果伊芙获得了交易过程中的信息，她也不能确定佩吉的密码。实际上，由于协议的零知识特点，伊芙不能获得可以在将来的交易中重复使用的密码 s_1，...，s_k 的任何信息。

12.3 习　题

1. 考虑图 12.2 中的通道图表。假设通到中央大厅的 4 扇门都是锁着的，所以需要持钥匙才能进入，但是出来不需要钥匙。佩吉声称自己有打开其中一扇门的钥匙。设计一个零知识协议，通过它佩吉能向维克托证明她能进入大厅。但是，又不能让维克托知道她能打开哪扇门。

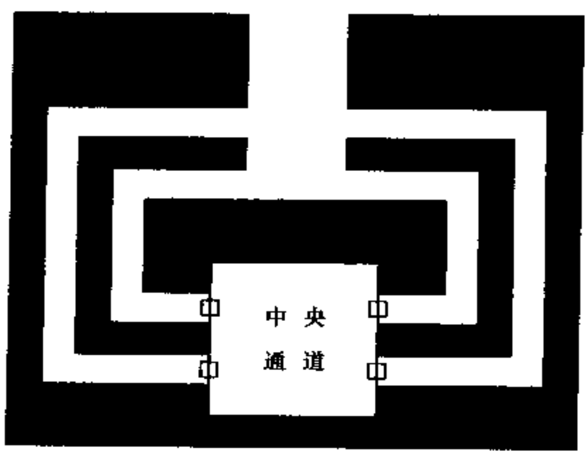


图 12.2　练习 1 的图

2. 假设 p 为一个大素数，α 为一个本原根，且 $\beta \equiv \alpha^a \pmod p$。数 p，α，β 是公开的。佩吉想向维克托证明她知道 a，但不想暴露它。他们按如下步骤进行：

(1) 佩吉选择一个随机数 $r \pmod{p-1}$。

(2) 佩吉计算 $h_1 \equiv \alpha^r \pmod p$ 和 $h_2 \equiv \alpha^{a-r} \pmod p$，并把 h_1 和 h_2 发给维克托。

(3) 维克托令 $i=1$ 或 $i=2$，并让佩吉传送 $r_1 = r$ 或 $r_2 = a - r \pmod{p-1}$。

(4) 维克托检验 $h_1 h_2 \equiv \beta \pmod p$ 和 $h_i \equiv \alpha^{r_i} \pmod p$。

(5) 重复上述步骤几次。

(a) 假设佩吉不知道 a，为什么她有时不能提供令维克托信服的数字？

(b) 如果他们重复上述步骤 t 次，并且佩吉不知道 a，那么佩吉能让维克托相信她知道 a 的概率为多少？

(c) 假定纳尔逊尝试另一种方法。他想让维克托相信他知道 a，所以他像以前一样选了一个随机数 r，但是并未传送 h_1，h_2。维克托让其提供 r_1，纳尔逊传送它。他们重复以上操作几次。为什么维克托不能确认任何事情？纳尔逊和佩吉的方案中哪些本质上的不同导致了这种情况？

3. 纳尔逊认为他理解零知识协议。他想向维克托证明他知道 n 的因数分解（等于 pq，p 和 q 为大素数），但并不想把因数分解透露给维克托和其他人。纳尔逊设计了以下方案：维克托选择一个随机整数 x 模 n，计算出 $y \equiv x^2 \pmod n$，并把 y 发给纳尔逊。纳尔逊算出 y 模 n 的一个平方根 s，并把它发给维克托。维克托检验是否 $s^2 \equiv y \pmod n$。维克托重复这个过程 20 次。

(a) 描述一下纳尔逊如何计算 s，你可以假定 p 和 $q \equiv 3 \pmod 4$。

(b) 解释维克托是如何运用这一程序高效发现 n 的因数分解的（因此，这并不是一个零知识协议）。

(c) 假定伊芙正在偷听，并知道了每一个 y 和 s 的值。伊芙是否可能获得有用的信息？（假定 y 值不能重复。）

4. 练习 2 给出了关于佩吉知道一个离散对数的零知识证明。这里是另外一种办法。假定 p 为一个大素数，α 为一个本原根，且 $\beta \equiv \alpha^a \pmod p$。数 p，α，β 是公开的。佩吉想向维克托证明她知道 a，但不想暴露它。做法如下：

(1) 艾丽斯选择一个随机整数 k，满足 $1 \le k < p-1$。计算 $\gamma \equiv \alpha^k \pmod p$，并把 γ 发给维克托。

(2) 维克托选择一个随机整数 r，满足 $1 \le r < p-1$，把 r 发给佩吉。

(3) 佩吉计算 $y \equiv k - ar \pmod{p-1}$，把 y 发给维克托。

(4) 维克托检验 $\gamma \equiv \alpha^y \beta^r \pmod p$ 是否成立。如果成立，就可以确认佩吉知道 a。

(a) 如果整个过程正确执行的话，证明验证方程式存在。

(b) 维克托获得了能计算出 a 的一些信息吗？

(c) 假定伊芙发现了 γ，r 和 y 的值，她能推算出 a 吗？

(d) 假定佩吉用同样的 k 值重复这个过程，而维克托用不同的 r_1 和 r_2 值，那么已获取佩吉和维克托之间所有交流信息的伊芙如何计算出 a？

上述程序是 **Schnorr 识别方案（Schnorr identification Scheme）**的基础。维克托可以是银行，而 a 可以是佩吉的个人识别密码。银行存有 β，佩吉必须证明她知道 a 才能访问她的

账户。同样，维克托可以是一个中央计算机，而佩吉可能正通过不安全的电话线登录到计算机上。佩吉的密码为 a，而中央计算机存储 β。

在 Schnorr 方案中，p 的选择通常要满足 $p-1$ 有一个大的素数因子 q，而 α 要求满足 $\alpha^q\equiv 1 \pmod p$ 来代替 α 为一个本原根的要求。因此相应地，y 的定义要求对 q 取模。而且 r 被要求满足 $1\leqslant r\leqslant 2^t$，$t$ 为某个值，比如 $t=40$。

第13章 密钥建立协议

至此，本书已经讨论了各种加密的概念，并集中讨论了不断发展的有关安全通信的算法。但是加密算法与其密钥的安全性息息相关。如果艾丽斯在开始一个与鲍勃的 DES 会话前向所有人公布了她的密钥，那么任何人都可以进行窃听。当然，这是一个荒谬的想法。但是，它反映出一个很重要问题的极端情况：如果艾丽斯和鲍勃不能通过相见来交换他们的密钥，那么他们是否仍然可以在不危及将来的通信安全的前提下确定一个密钥？

在本章中我们将讨论关于密钥建立协议的重要话题。我们尤其考虑了为了制定对称密钥而分享秘密信息所带来的基本问题。我们所说的对称密钥是指一种诸如 DES 的体制，在这种体制里发送者和接收者用相同的密钥。这与诸如 RSA 的公开密钥方法相反，在 RSA 中发送者有一个密钥（加密指数），接收者有另一个密钥（解密指数）。

在密钥建立协议中，艾丽斯和鲍勃之间需要进行一定顺序的几个步骤，以便分享某些在制定密钥时需要的秘密信息。由于公开密钥加密方法需要用到存储在公共数据库中的公开编码密钥，我们可能会认为公开密钥加密提供了解决这个问题的简单方法。这只对了一部分。公开密钥加密的主要缺点在于，当与最好的对称密钥方法进行比较时，即使最好的公开密钥加密计算也是比较慢的。例如，RSA 要求幂运算，这就没有在 DES 中进行的位混合快。因此，有时 RSA 用来传送一个 DES 密钥，该密钥将被用来传送大量的数据。然而，一个需要在短时间内与许多客户机通信的中心服务器需要的密钥建立方法有时比目前的公开密钥算法快。因此，在这种情况和其他各种不同的情况下，我们就有必要考虑其他的方法来交换和制定对称加密算法的密钥。

密钥的制定有两种基本的类型。在密钥协商协议中，双方预先都不知道密钥，该密钥由双方交互的结果决定。在密钥分发协议中，一方确定了一个密钥后将其发送给另一方。

13.1 密钥协商协议

密钥协商是一种协议，依据这种协议艾丽斯和鲍勃双方之间通过交换信息来制定一个密钥。任一方都从一个交换信息的函数计算得到密钥。

事实证明，密钥协商协议用于非对称（公开密钥）加密术时效果最好。我们将讲述一个著名的源于 Diffie-Hellman 的协议，该协议提供了含有两个信息传递方的密钥的制定方法。正如我们后面将讨论的，这种方法没有提供身份鉴别，为了弥补这个缺陷，我们可以应用数字签名方案。这种更精密的协议被称为**站对站**（STS，station-to-station）协议。

Diffie-Hellman 密钥交换

为了通信艾丽斯和鲍勃想制定一个密钥，用 Diffie-Hellmen 方案实现的过程如下：

1. 艾丽斯或鲍勃选择一个安全的大素数 p 和一个本原根 $\alpha \pmod p$，这两个数都公开。

2. 艾丽斯选定一个保密的随机数 x，满足 $1 \leq x \leq p-2$，同时鲍勃选定一个保密的随机数 y，满足 $1 \leq y \leq p-2$。

3. 艾丽斯发送 $\alpha^x \pmod p$ 给鲍勃，同时鲍勃发送 $\alpha^y \pmod p$ 给艾丽斯。

4. 利用各自收到的消息，他们可以计算出会话密钥 K。艾丽斯通过 $K \equiv (\alpha^y)^x \pmod p$ 计算 K 的值，而鲍勃通过 $K \equiv (\alpha^x)^y \pmod p$ 计算 K 的值。

中间人（Intruder-in-the-Middle）攻击

伊芙最近刚知道了马和车的区别，她声称可以同时与两个象棋大师下两局棋，而且或者赢一局或者两局都是平局。她的策略很简单，她等到第一个大师出棋后，在与第二个大师下棋时走相同的一步棋。当第二个大师回招后伊芙又用这一招来应对第一个大师。照这样下去，伊芙不可能输掉两局棋（除非她在传递某步棋时由于稍微的拖延而遇到时间问题）。

一种与之类似的策略叫做中间人攻击，可以用来应对前面提到的 Diffie-Hellman 方案。该策略的步骤如下：

1. 伊芙选定一个指数 z。

2. 伊芙劫取 α^x 和 α^y。

3. 伊芙将 α^z 发送给艾丽斯和鲍勃（这时艾丽斯相信其所接收到的是 α^y，而鲍勃也相信其所接收到的是 α^x）。

4. 伊芙计算 $K_{AO} \equiv (\alpha^x)^z \pmod p$ 和 $K_{OB} \equiv (\alpha^y)^z \pmod p$。艾丽斯并未意识到伊芙在中间，所以也计算 K_{AO} 的值，鲍勃也计算 K_{OB} 的值。

5. 当艾丽斯将用 K_{AO} 加密后的一条消息发送给鲍勃时，伊芙截取该消息并进行解码，用 K_{OB} 加密后发送给鲍勃。鲍勃用 K_{OB} 解密而得到该消息。鲍勃没有理由相信这种通信是不安全的。同时，伊芙正在看她得到的这些有趣的信息。

站对站协议（Station-to-Station）

为了避免这种中间人攻击，人们需要一个程序在密钥形成时对艾丽斯和鲍勃相互间的身份进行鉴别。一个能实现这个目的的协议被认为是经鉴别的密钥协商协议。

这种标准的解决方案采用了数字签名。每个用户 U 有一个含认证算法 ver_U 的数字签名函数 sig_U。例如，sig_U 能产生一个 RSA 或 ElGamal 签名，ver_U 检查该值是否为 U 的合法签名。认证算法由一个可信任的权威人士特伦特编译并公开，特伦特保证 ver_U 确实是为用户 U 而定的认证算法，不是为伊芙而定的。

现在假设艾丽斯和鲍勃需要制定一个用于加密函数 E_K 的密钥。他们按照 Diffie-Hellman 密钥交换方案进行操作，但加入了如下数字签名的特点：

1. 他们选定一个大素数 p 和一个本原根 α。

2. 艾丽斯选定一个随机数 x，鲍勃选定另一个随机数 y。

3. 艾丽斯计算出 $\alpha^x \pmod p$，鲍勃计算出 $\alpha^y \pmod p$。

4. 艾丽斯将 α^x 发送给鲍勃。

5. 鲍勃计算出 $K \equiv (\alpha^x)^y \pmod p$。

6. 鲍勃将 α^y 和 $E_K(sig_B(\alpha^y, \alpha^x))$ 发送给艾丽斯。

7. 艾丽斯计算出 $K \equiv (\alpha^y)^x \pmod p$ 。

8. 艾丽斯解密 $E_K(sig_B(\alpha^y,\alpha^x))$，得到 $sig_B(\alpha^y,\alpha^x)$ 。

9. 艾丽斯请求特伦特验证 ver_B 是否为鲍勃的验证算法。

10. 艾丽斯用 ver_B 来验证鲍勃的签名。

11. 艾丽斯发送 $E_K(sig_A(\alpha^x,\alpha^y))$ 给鲍勃。

12. 鲍勃解密后，要求特伦特验证 ver_A 是否为艾丽斯的验证算法，然后用 ver_A 验证艾丽斯的签名。

Diffie，van Oorschot 和 Wiener 制定的该方案的一个增强版本被称为站对站协议。注意：由于一个不正确的密钥经解密后得到一个合法签名的可能性很小，所以艾丽斯和鲍勃也确信他们所用的是同一个密钥。

13.2　密钥预分发

在该协议的一个最简单的形式中，如果艾丽斯需要与鲍勃通信，其密钥或密钥表（一个描述什么时候用什么密钥的列表）是预先确定的，并以某种方式使该信息在二者之间安全地传送。例如，二战时期德国海军就使用了该方法。然而，英国利用从其俘获的战船上找到的密码本发现了德军每天的密钥，从而获取了德军的信息。

预先分配协议有一些明显的限制和缺陷。首先，它要求艾丽斯和鲍勃双方已经预先接触或预先在二者间建立了一个安全的通道。其次，一旦艾丽斯和鲍勃碰面并交换了信息，那么在密钥安全有危险时，除非他们再次碰面，否则将无法交换密钥信息。

这里有一个稍作改变的普遍的情况。首先，我们需要一个可信的权威人士，我们称他为特伦特。对任一用户对，称为（A，B），特伦特提出一个将在对称加密方法中用作密钥的随机密钥 K_{AB}（因此 $K_{AB}=K_{BA}$）。假设特伦特有足够的能力，并已经预先为每个用户建立了一个安全通道，他为他的用户分配已确定的所有密钥。这样一来，如果特伦特对 n 个用户负责，那么每个用户将接收到 $n-1$ 个密钥来存储，而特伦特也必须安全地发送 $n(n-1)/2$ 个密钥。如果 n 很大，这将是一个问题。每个用户所要求的存储量也是一个问题。

减少可信权威人士所发送的信息量的一种方法是**布洛姆密钥预先分配方案**（Blom key pre-distribution scheme）。假定一个网络有 n 个用户，设 p 为一个大素数，且 $p \geq n$。每个用户都知道该素数 p。该协议操作如下：

1. 给网络中的每个用户 U 分配一个不同的公开数 $r_U \pmod p$ 。

2. 特伦特选定 3 个保密的模 p 随机数 a，b，c。

3. 对每个用户 U，特伦特计算出

$$a_U \equiv a + br_U \pmod p \qquad b_U \equiv b + cr_U \pmod p$$

并通过安全通道将其发送给用户 U。

4. 每个用户 U 产生下面的线性多项式：

$$g_U(x) = a_U + b_U x。$$

5. 如果艾丽斯（A）需要与鲍勃（B）通信，艾丽斯算出 $K_{AB} = g_A(r_B)$，而鲍勃算出 $K_{BA} = g_B(r_A)$。

6. 可以证明 $K_{AB}=K_{BA}$（见练习 2）。艾丽斯和鲍勃通过一个诸如 DES 的对称加密体制，利用该密钥 K_{AB}（或是源自 K_{AB}的一个密钥）进行通信。

例：考虑一个有 3 个用户艾丽斯、鲍勃和查尔斯的网络。设 $p=23$，并令

$$r_A = 11,\ r_B = 3,\ r_C = 2。$$

假定特伦特选定的数为 $a=8$，$b=3$，$c=1$，则对应的线性多项式为：

$$g_A(x) = 18 + 14x,\ g_B(x) = 17 + 6x,\ g_C(x) = 14 + 5x。$$

现在就可以算出这种方案产生的密钥：

$$K_{AB} = g_A(r_B) = 14,\ K_{AC} = g_A(r_C) = 0,\ K_{BC} = g_B(r_C) = 6。$$

在这个例子中，我们很容易验证 $K_{AB}=K_{BA}$是否成立。 ■

如果有两个用户伊芙和奥斯卡联合起来，他们可以确定 a，b 和 c，这样就能找到所有用户的所有数字 a_A，b_A。他们按照如下的步骤进行。他们知道数字 a_E，b_E，a_O，b_O。为这些数字的最后三位定义的方程式可以用下面的矩阵表示：

$$\begin{pmatrix} 0 & 1 & r_E \\ 1 & r_O & 0 \\ 0 & 1 & r_O \end{pmatrix}\begin{pmatrix} a \\ b \\ c \end{pmatrix} \equiv \begin{pmatrix} b_E \\ a_O \\ b_O \end{pmatrix} \pmod p$$

矩阵最后三行的行列式值是 r_E-r_O。由于数字 r_A 模 p 的选择是惟一的，行列式的值模 p 是非零的，所以体制有惟一的解 a，b，c。

如果没有伊芙的帮助，奥斯卡得到的仅仅是一个 2×3 阶的矩阵，因而无法解得 a，b，c。实际上，假定他想要算出艾丽斯和鲍勃所用的密钥 K_{AB}，那么由于 $K_{AB}\equiv a+b(r_A+r_B)+c(r_Ar_B)$（见练习 2），奥斯卡得到的矩阵方程为：

$$\begin{pmatrix} 1 & r_A + r_B & r_Ar_B \\ 1 & r_O & 0 \\ 0 & 1 & r_O \end{pmatrix}\begin{pmatrix} a \\ b \\ c \end{pmatrix} \equiv \begin{pmatrix} K_{AB} \\ a_O \\ b_O \end{pmatrix} \pmod p$$

该矩阵的行列式值 $(r_O-r_A)(r_O-r_B)\not\equiv 0 \pmod p$。因此，对所有可能的 K_{AB}值该矩阵都有解 a，b，c。这就意味着奥斯卡得不到有关 K_{AB}的任何信息。

对每个 $k\geq1$，都有可以防范 k 个用户合并的布洛姆方案，但是没法防范 $k+1$ 个用户的联合［参考 Blom］。

13.3 密钥分发

密钥预分发协议有个缺点，即密钥是预先决定的且在某段时间后没有简单的方法来改变该密钥。当长时期使用同一密钥时，用户密钥的安全就会面临危险。传递的信息越多，就会有更多的数据被用来准备统计攻击。

在这节中，我们将讨论一类称为**传送协议**（transport protocols）的密钥建立协议。在这种协议中，要么艾丽斯决定一个密钥而后发送给鲍勃，要么由可信的权威人士特伦特作为一个密钥服务器。在第一种情况下，艾丽斯为了把密钥传给鲍勃必须使用某种安全协议。在第二种情况下，当艾丽斯想要与鲍勃通信时，她从特伦特那里请求一个适合单一会话的密钥，

然后特伦特通过某个安全协议把这个会话密钥传送给艾丽斯和鲍勃。

Shamir 的三步协议

艾丽斯想要通过一个建立在公共通道上的通信来传递密钥 K 给鲍勃。首先，艾丽斯选择一个安全（也就是像离散对数问题那样模 p 是很难的）并且足够大的大的素数 p 来表示密钥 K。例如，如果艾丽斯正准备发送一个 56 比特的 DES 密钥，她就需要选择一个至少 56 比特长的素数。然而，出于安全的考虑，她会选择一个比 56 比特长得多的素数。艾丽斯发布 p，使鲍勃（或其他任何人）可以下载它。鲍勃下载 p 后，艾丽斯和鲍勃按如下步骤进行：

1. 艾丽斯选择一个满足 $\gcd(a, p-1)=1$ 的随机数 a，鲍勃选择一个满足方程 $\gcd(b, p-1)=1$ 的随机数 b。我们将用 a^{-1} 和 b^{-1} 表示 a 和 $b \pmod{p-1}$ 的逆。
2. 艾丽斯发送 $K_1 \equiv K^a \pmod p$ 给鲍勃。
3. 鲍勃发送 $K_2 \equiv K_1^b \pmod p$ 给艾丽斯。
4. 艾丽斯发送 $K_3 \equiv K_2^{a^{-1}} \pmod p$ 给鲍勃。
5. 鲍勃算出 $K \equiv K_3^{b^{-1}} \pmod p$。

在该协议的最后，艾丽斯和鲍勃都得到了密钥 K。该协议的安全性在于离散对数问题的难解性。然而，如果没有附加的安全防护措施，这个过程不能抵御中间人攻击。

Kerberos

Kerberos（由古希腊神话中一只守护地狱入口的三头狗而得名）是一个对称加密协议的真实实现，该协议的目的是为网络中各用户间的密钥交换提供高等级的认证和安全性。这里我们不严格地使用术语 *users*，因为一个用户可能是一个人，也可能是一个需要与另一程序通信的程序。Kerberos 是从 M. I. T 中一个更大的称为 Athena 的发展项目中产生的。Athena 项目的目的是将一个巨大的电脑工作站网络集成在 M. I. T 的一个大学生团体的课程中。该课程允许学生在网络的任何地方轻易地访问他们的文件。有人可能会猜想，这样的一个发展会很快导致网络安全方面的问题。尤其是，通过诸如 Athena 这样的公开网络进行通信是很不安全的，而且可能很容易通过网络查看数据流并寻找诸如密码和某些用户想要保密的其他类型的令人感兴趣的信息位。为了处理这一安全性问题，Kerberos 被提出来了。接下来我们讲述 Kerberos 的基本模型，并描述其概念及作用。想要了解更加全面的描述，请参阅 [Schneier]。

Kerberos 基于一个客户端/服务器的结构。客户端不是一个用户就是一个需要完成某种任务的软件程序。例如，某一客户端可能想要发 e-mail、打印文档或是加载外设。服务器是一个较大的实体，其作用是为客户端提供服务。例如，在 Internet 和 WWW 上有一个域名服务器的概念，其作用是为诸如 e-mail 程序或 Internet 浏览器提供域名或地址。

Kerberos 的基本模型有如下参与者：

克利夫：一个客户

瑟尔吉：一个服务器

特伦特：一个可信的权威机构

格兰特：一个授予许可证的服务器

可信权威机构也被称为一个认证服务器。开始时，克利夫和瑟尔吉之间没有共享的密钥信息，而 Kerberos 的目的在于将信息安全地传送给他们。Kerberos 协议的一个结果是瑟尔吉将确认克利夫的身份（他不想与一个假的克利夫进行通信，不是吗?），并且一个会话密钥

将被建立起来。

如图 13.1 所描述，该协议中，首先克利夫向特伦特申请一个许可服务的许可证。由于特伦特是一个强大的可信权威机构，所以他有一个包含所有客户的密钥信息的数据库（由于这个原因，特伦特有时也被认为是 Kerberos 服务器）。特伦特返回一个用该客户的密码信息加密的许可证。这时克利夫可能想使用瑟尔吉提供的服务，但是在这之前，他必须被允许与瑟尔吉通话。克利夫将其许可证出示给格兰特——授予许可证的服务器。格兰特接收到该许可证后，如果一切都正常（注意该许可证含有一些克利夫的认证信息），那么格兰特给克利夫一个新的许可证，该许可证将允许克利夫使用瑟尔吉所提供的服务（而且仅限于瑟尔吉的服务；该许可证对另一个服务器 Sarah 无效）。现在克利夫有了一个可以向瑟尔吉出示的服务许可证，他将该许可证和认证证书发送给瑟尔吉。瑟尔吉用认证证书检查该许可证，以确保它是合法的。如果这个最终的信息交换合格，瑟尔吉将为克利夫提供服务。

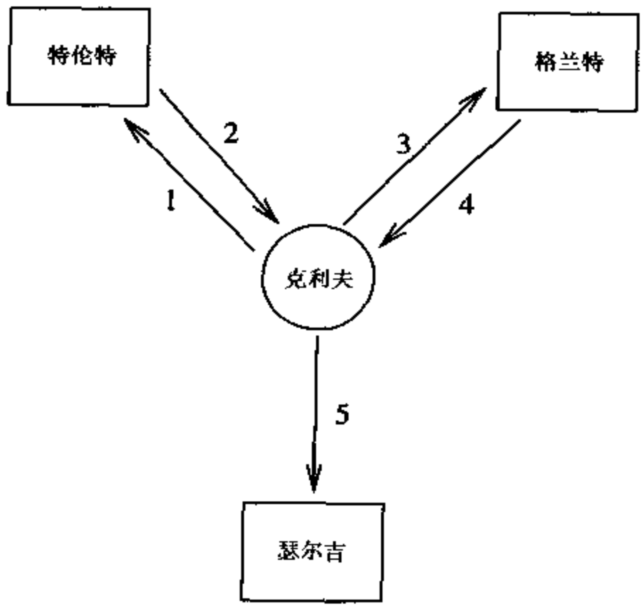


图 13.1 Kerberos

Kerberos 协议是我们每天都会用到的一个正式版本的协议（例如，在银行兑取现金或去儿童乐园里骑马）。

现在我们更深入地看看 Kerberos 协议。Kerberos 应用了一个对称加密算法。在第五版中，Kerberos 运用了以 CBC 模式进行操作的 DES；然而，对任何对称加密算法都是适合的。

1. 克利夫对特伦特：克利夫发送一条消息给特伦特，其中包含他的姓名和他将要用到的授予许可证的服务器名称（这里是指格兰特）。

2. 特伦特对克利夫：特伦特在其数据库中查找克利夫的名字，如果找到了，他将产生一个在克利夫和格兰特间使用的会话密钥 K_{CG}。特伦特也有一个密钥 K_C，他可以凭该密钥与克利夫进行通信，于是他用该密钥加密克利夫-格兰特的会话密钥：

$$T = e_{K_C}(K_{CG})。$$

另外，特伦特产生一个允许克利夫向格兰特认证自己的票证许可票证（TGT），该许可证是用格兰特的私人密钥 K_G（特伦特也有这个密钥）进行加密的：

$$TGT = \text{格兰特的名字} \parallel e_{K_G}(\text{克利夫的名字,克利夫的地址,时间戳 1}, K_{CG})。$$

在这里 ‖ 用来表示并集。克利夫接收到的许可证是这两个子许可证的并集：

$$\text{Ticket} = T \parallel TGT。$$

3. 克利夫对格兰特：克利夫可以由密钥 K_C 求出他与特伦特共享的密钥 K_{CG}。现在克利夫就可以利用 K_{CG} 安全地与格兰特进行通信了。这时克利夫这时产生由其名称、地址和一个时间戳构成的认证码。他用 K_{CG} 对它进行加密，得到

$$\text{Auth}_{CG} = e_{K_{CG}}(\text{克利夫的名称,克利夫的地址,时间戳 2})。$$

这时克利夫将 Auth_{CG} 和 TGT 发送给格兰特，以便格兰特可以管理这一服务许可证。

4. 格兰特对克利夫：现在格兰特得到了 Auth_{CG} 和 TGT。由于 TGT 的部分是用格兰特的密码进行加密的，所以格兰特可以提取该部分然后将其解密。这样他就可以恢复克利夫的名称、地址、时间戳 1 和 K_{CG}。这时格兰特可以用 K_{CG} 解出 Auth_{CG} 以核实克利夫请求的正确性。也就是说，$d_{K_{CG}}(\text{Auth}_{CG})$ 将提供另一个含有克利夫名称、地址和一个不同的时间戳的副本。如果这两个版本的克利夫名称、地址匹配而且时间戳 1 和时间戳 2 彼此十分接近，格兰特将宣布克利夫为合法的。这时克利夫已经得到格兰特的认可，格兰特会为克利夫产生一个会话密钥 K_{CS} 以便与瑟尔吉进行通信，同时也将返回一个服务许可证。格兰特有一个与瑟尔吉共享的密钥 K_S。返回的服务许可证为：

$$\text{ServTicket} = e_{K_S}(\text{克利夫的名称,克利夫的地址,时间戳 3,到期时间}, Kcs)。$$

这里的到期时间是一个用来描述该服务许可证的有效期限的量。这里的会话密钥是用克利夫和格兰特间的一个会话密钥加密的：

$$e_{K_{CG}}(K_{CS})。$$

然后，格兰特将 ServTicket 和 $e_{K_{CG}}$（K_{CS}）发送给克利夫。

5. 克利夫对瑟尔吉：现在克利夫已经做好了开始利用瑟尔吉提供的服务的准备。他首先将解密 $e_{K_{CG}}$（K_{CS}）以得到会话密钥 K_{CS}，在与瑟尔吉进行通信时将用到该密钥。然后，克利夫产生一个与瑟尔吉公用的认证码：

$$\text{Auth}_{CS} = e_{K_{CS}}(\text{克利夫的名称,克利夫的地址,时间戳 4})。$$

这时克利夫将 Auth_{CS} 和 ServTicket 发送给瑟尔吉。瑟尔吉可以解密 ServTicket 并从中提取出与克利夫公用的会话密钥 K_{CS}。利用该会话密钥瑟尔吉可以解密出 Auth_{CS}，核实克利夫是否就是其人，并检验时间戳 4 是否在时间戳 3 的到期时间内。如果时间戳 4 不在时间戳 3 的到期时间内，则克利夫的许可证过期，瑟尔吉将拒绝克利夫的服务请求。否则，克利夫和瑟尔吉就可以通过 K_{CS} 进行信息交换。

13.4　公钥基础设施（PKI）

公钥加密法是一个涉及身份鉴别、密钥分发和反拒绝的强大工具。在这些应用中，公钥是公开的，但是当你访问公钥时，如何能保证艾丽斯的公钥确实是属于艾丽斯的呢？有可能伊芙已经用她自己的公钥替代了艾丽斯的公钥。除非在如何形成密钥和密钥的真实性和合法性认证中存在可靠性，否则，公钥加密法的优点就太小了。

为了使公钥加密法能在商业应用中发挥作用，有必要加上一个基础设施来跟踪公钥。公

钥基础设施也可以简称为 PKI，是一个框架结构，它是由定义加密体制运转规则的策略和用于产生和发布密钥和证书的程序构成的。

所有的 PKI 都是由证书授权和合法性操作组成的。证书授权将一个公钥绑定给一个实体，例如一个用户或一条信息。合法性确保了证书是合法的。

证书（**certificate**）是指由其发布者签署的大量信息，这里发布者一般指的是**证书认证中心**（**CA**）。有很多类型的证书。两种较流行的是身份证书和验证身份证书。身份证书包含了一个实体的身份信息，诸如 e-mail 地址和该实体的公钥列表等。验证身份证书包含了描述访问权限的信息。在任何一种情况下，数据都用 CA 的私人密钥进行特别加密。

假设我们有一个 PKI，且 CA 发布了艾丽斯和鲍勃的身份证书。如果艾丽斯知道了 CA 的公钥，那么她就可以获取已发布的鲍勃的加密身份证书，并提取出鲍勃的身份信息和与鲍勃进行安全通信所需的公钥列表。这种情况与传统公钥的区别在于鲍勃不用发布自己的密钥，而代之以艾丽斯和发布者间的信托关系。艾丽斯可能不会像相信诸如政府或电话公司这样的 CA 一样相信鲍勃。信任的概念是 PKI 的关键，可能是 PKI 最重要的特性之一。

一个单独的实体能够跟踪并发布每个 Internet 用户的公钥是不可能的。相反，PKI 通常是由多个 CA 和他们发布的证书组成的，这些 CA 可以相互认证身份。这样鲍勃就可能跟一个不同于艾丽斯的 CA 相关联，当请求鲍勃的身份认证时，艾丽斯可能仅当她的 CA 信任鲍勃的 CA 时才会相信该认证。在像 Internet 这样的大型网络中，艾丽斯和鲍勃之间可能会有很多 CA，这样在二者间的每个 CA 相互信任就很必要了。

另外，大多数 PKI 有不同的信任等级，以允许某些 CA 以不同的信任度认证其他 CA。有可能某些 CA 仅信任其他 CA 完成特定的任务。例如，艾丽斯的 CA 可能仅委托鲍勃的 CA 认证鲍勃的身份而不是其他的 CA，而艾丽斯的 CA 则信任 Dave 的 CA 来认证其他的 CA。信托关系可以变得非常精细，随着这种关系变得越来越复杂，艾丽斯对其收到的证书的信任度就变得很难确定。

这里有两个 PKI 在实际应用中的实例。

Pretty Good Privacy（PGP）是最初由 Phil Zimmerman 制作来对 e-mail 进行加密和签名的一个程序。它是采用公钥和私钥加密法的结合来实现的。每名用户保留了一个由与该用户交换 e-mail 的其他人的公钥组成的密钥环。为了保护艾丽斯的密钥环不被篡改，她用其私钥来签名她的密钥环中的密钥。

这种 PGP 程序允许用户交换密钥环，当艾丽斯收到鲍勃的密钥环时，她授予其如下的四种信任程度之一：

- 完全信任：艾丽斯完全信任鲍勃并且她会信任任何用鲍勃的密钥签名的密钥。
- 部分信任：艾丽斯在某种程度上信任鲍勃但不是完全信任，在她信任一个被鲍勃签名的密钥前，她要求该密钥已经由其他的个体签名。
- 不信任：艾丽斯不信任鲍勃，因而也不信任任何用鲍勃的密钥签名的密钥证书。
- 不确定：艾丽斯不能确定是否应该信任鲍勃。典型地，这被视为与不信任是相同的。

随着用户共享密钥环，用户间的**信任网**（**web of trust**）就被建立起来了。有可能艾丽斯最终会信任某个她从未见过的用户。PGP 的这种简单性使之在诸如安全 e-mail 等应用中很受欢迎。然而，对于电子商务的应用，更复杂的 PKI 就是必需的了。

X.509 就是 PKI 中的一种。它是一种国际标准，被设计来为大型计算机网络上的目录服

务提供认证。由于它本身是作为国际标准化组织和国际电信联盟的一个标准，所以很多产品都是基于它开发出来的。例如，X.509 被用在签证和万事达信用卡的安全电子交易的标准中。

X.509 证书的组成之一是用来识别用户和证书的发行者的命名约定。在 X.509 的第一版和第二版中，该标准应用了一个称为 X.500 的命名约定。在第三版中考虑到了可选择的命名约定。另外，X.509 证书包含了用于指定该证书遵守 X.509 的哪一个版本的域、用于描述该证书有效时间的合法期限以及用户的公钥信息。X.509 证书的第三版允许 CA 在证书中包含一系列信任策略。这些信任策略可以告诉用户使用已认证的密钥的目的，也可以规定某一特定的 CA 可以对哪一组用户进行证书认证。

由于 X.509 证书包含了描述信任策略的域，它比 PGP 更加强大，可以用在需要更多特定的证书等级的应用中。例如，用 X.509 可以指出一个公钥仅适合于安全的 e-mail 应用而不适合于电子商务应用。

13.5 习　题

1. 在有 3 个用户 A，B，C 的网络中，我们想要用布洛姆方案来制定各对用户间的会话密钥。令 $p=31$ 且

$$r_A = 11 \qquad r_B = 3 \qquad r_C = 2。$$

假设特伦特选定的数字为

$$a = 8 \qquad b = 3 \qquad c1 = 1。$$

计算出会话密钥。

2. (a) 证明在布洛姆方案中，$K_{AB} \equiv a + b(r_A + r_B) + cr_Ar_B \pmod p$。

(b) 证明 $K_{AB} = K_{BA}$。

(c) 另一个观察布洛姆方案的方法是利用一个二元多项式。定义该多项式为 $f(x,y) = a + b(x+y) + cxy \pmod p$。根据 f 表示密钥 K_{AB}。

3. 你（U）和我（I）是一个网络上的恶意用户，我们应用布洛姆方案来制定密钥且 $k=1$。我们决定一起计算出该网络中的其他会话密钥。特别地，假设 $p=31$，$r_U=9$，$r_I=2$。我们已经从可信权威机构特伦特处接收到 $a_U=18$，$b_U=29$，$a_I=24$，$b_I=13$。计算出 a，b 和 c。

4. 这里是中间人攻击的另一版本。它有这样的优点，即伊芙不需要截取和转发鲍勃和艾丽斯之间的所有消息。假设伊芙发现 $p=Mq+1$，其中 p 为一素数，M 是一个很小的数。与前面相同，伊芙截取 α^x 和 α^y，然后向鲍勃发送 $(\alpha^x)^q \pmod p$，向艾丽斯发送 $(\alpha^y)^q \pmod p$。

(a) 证明艾丽斯和鲍勃各自算出的密钥相同。

(b) 证明对某一个 K，仅有 M 个可能值，所以伊芙可以通过穷尽搜索发现 K 值。

5. 鲍勃、特德、卡罗尔和艾丽斯想对一个共同密钥达成一致（也就是说密码密钥）。他们公开地选定一个大素数 p 和一个本原根 α。他们私下分别选定了随机数 b，t，c，a。描述一个协议，该协议允许他们安全地计算出 $K \equiv \alpha^{btca} \pmod p$（忽略中间人攻击）。

6. 假设天真的纳尔逊试图按如下步骤实现一个类似于 Diffie-Hellman 密钥交换的协议。纳尔逊想要发送密钥 K 给海蒂。他选择了一个一次一密密钥 K_N 并把它与 K 异或，然后他发送 $M_1 = K_N \oplus K$ 给海蒂。海蒂用她的一次一密密钥 K_H 与她接收到的数异或，得到 $M_2 = M_1 \oplus K_H$。海蒂发送 M_2 给纳尔逊，纳尔逊计算出 $M_3 = M_2 \oplus K_N$。纳尔逊将 M_3 发送给海蒂，海蒂从 $M_3 \oplus K_3$ 中恢复 K 值。

（a）证明 $K = M_3 \oplus K_3$。

（b）假设伊芙截取了 M_1，M_2，M_3，她怎样才能恢复 K 值？

信息论

第 14 章

本章我们将介绍一下密码体制安全性背后的理论概念。基本问题如下：如果伊芙观察到一段密文，那么她能够得到她本来不知道的关于这个加密密钥的任何新信息吗？为了处理这个问题，我们需要一个关于信息的精确定义。这包括概率和称为熵的一种重要的测量方法的使用。

本章中的许多观点来源于 20 世纪 40 年代后期的克劳德·香农。

开始前，让我们先看一个例子。掷一个标准的六面形骰子，A 表示点的数目为奇数这个事件，B 表示点的数目大于等于 3 这个事件。如果某人告诉你这次抛掷属于事件 $A\cap B$，那么你应该知道滚动的结果仅仅有两种可能性。在这种情况下，$A\cap B$ 告诉你的要比仅仅从事件 A 或事件 B 获得的关于抛掷后的结果的情况多得多，也就是说，包含在 $A\cap B$ 中的信息比仅仅包含在 A 或 B 中的信息要多。

信息的概念与不确定性的概念是密切联系在一起的，回到掷骰子的例子，如果已经知道 $A\cap B$ 事件发生了，那么对于抛掷后的值比在仅仅知道事件 A 发生的情况下有更少的不确定性。因此，当不确定性降低时信息反而增加了。熵提供了一种信息增加或实验结果提供的不确定性减少的测量方法。

14.1 概率回顾

本节我们将简单地介绍一下随后要用到的概率的概念，对概率及其产生的各种特性的理解对熵的研究是必要的。

考虑一个实验 X，其可能的结果在一个集合 χ 中。例如，X 可能是抛一枚硬币，输出 $\chi=\{$正面，反面$\}$。假设给每一个输出都分配了一个概率，在本例中，p（$X=$正面）$=1/2$，p（$X=$反面）$=1/2$。通常一个实验的结果 X 被称为**随机变量（random variable）**。

一般地，对每一个 $x\in\chi$，定义 $X=x$ 的概率为

$$p_X(x)=p_x=p(X=x)。$$

注意到 $\sum_{x\in\chi}p_x=1$，如果 $A\subseteq\chi$，令

$$p(A)=\sum_{x\in A}p_x,$$

这是 X 在 A 中取某个值的概率。

通常进行的实验都是在测量几个不同的事件，这些事件可能是相关的也可能是不相关

的，但是它们都可以集中到一起来形成一个新的随机事件。例如，如果有两个随机事件 X 和 Y，它们可能的结果分别为 χ 和 $\mathcal{Y}$，那么可以产生一个把这两个事件聚合到一起的新随机事件 $Z=(X,Y)$，在这种情况下，新事件 Z 有一个可能的输出集合 $\mathcal{Z}=\chi\times\mathcal{Y}$，事件 Z 有时被称为联合随机变量。

例：从一副标准的扑克中抽取一张牌。令 X 表示这张纸牌的种类，所以 $\chi=\{$梅花，方片，红桃，黑桃$\}$，令 Y 表示这张牌的值，所以 $\mathcal{Y}=\{2,3,\cdots,A\}$，那么 $\mathcal{Z}$ 就给出了这张牌的 52 种可能性。注意到如果 $x\in\chi$，$y\in\mathcal{Y}$，那么 $p((X,Y)=(x,y))=p(X=x,Y=y)$ 就仅仅表示被抽中的纸牌类别为 x 且值为 y 的概率。因为所有的纸牌是相同的概率，所以这个概率为 1/52，这等于 $X=x$ 的概率（也就是 1/4）乘以 $Y=y$ 的概率（也就是 1/13）。正如我们后面要讨论的，这说明 X 和 Y 是相互独立的。 ■

例：投掷一个骰子。假设我们只对两种情况感兴趣：点的数目是否为奇数，是否大于等于 2。如果点的数目是偶数，令 $X=0$；如果点的数目是奇数，令 $X=1$；如果点的数目小于 2，令 $Y=0$；如果点的数目大于等于 2，令 $Y=1$，那么 $Z=(X,Y)$ 就一起给出了我们两个实验的结果。注意点的数目为奇数且小于 2 的概率为 $p(Z=(1,0))=1/6$，不是 $p(X=0)\cdot p(Y=0)=(1/2)(1/6)=1/12$，这意味着 X 和 Y 不是相互独立的。正如我们所看到的，这与 X 为我们所提供的关于 Y 的信息的事实是密切相关的。 ■

定义

$$p_{X,Y}(x,y)=p(X=x,Y=y)。$$

注意，我们可以改变 $X=x$ 的概率为

$$p_X(x)=\sum_{y\in\mathcal{Y}}p_{X,Y}(x,y)。$$

所以如果对于所有的 $x\in\chi$，$y\in\mathcal{Y}$，

$$p_{X,Y}(x,y)=p_X(x)p_Y(y)$$

成立，那么我们就说事件 X 和 Y 是**相互独立**的（**independent**）。在前面的例子中，纸牌的类别和纸牌的值是相互独立的。

我们也对 $X=x$ 事件发生的情况下 Y 的概率感兴趣，如果 $p_X(x)>0$，定义 $X=x$ 的情况下 $Y=y$ 的**条件概率**（**conditional probability**）为

$$p_Y(y\mid x)=\frac{p_{X,Y}(x,y)}{p_X(x)}。$$

考虑它的一种方法是对集合 $X=x$ 进行限制，就可得到总概率 $p_X(x)=\sum_y p_{X,Y}(x,y)$，$Y=y$ 的总和的分数是 $p_Y(y\mid x)$。

注意，当且仅当对于所有的 x,y，$p_Y(y\mid x)=p_Y(y)$ 成立时，X 和 Y 才是独立的，换句话说，X 发生变化对于 y 的概率是没有影响的。

有一个很好的办法可以实现从已知 X 时 Y 的条件概率到已知 Y 时 X 的条件概率的转换，这就是**贝叶斯定理**（**Bayes's Theorem**）。

贝叶斯定理：如果 $p_X(x)>0$ 和 $p_Y(y)>0$，那么

$$p_X(x\mid y)=\frac{p_X(x)p_Y(y\mid x)}{p_Y(y)}。$$

这个定理的证明可根据定义简单书写条件概率来完成。

14.2 熵

投掷一个六面的骰子和十面的骰子，哪一个的结果更难确定？如果你对每一次投掷的结果都进行猜测，似乎对十面骰子比六面骰子更容易猜错，因此，十面骰子有更大的不确定性。相似地，比较一下抛一枚正反面朝上概率相同的均匀的硬币与抛一枚正面朝上的概率为90%的硬币这两种情况，哪一种更不确定？显然是抛均匀的硬币不确定性大，同样是因为它的概率有更大的随机性。

在不确定性的定义中，我们想确认两个有相同概率分配的随机变量 X 和 Y 具有相同的不确定性，为了达到这个目的，不确定性的测量必须仅仅是概率分配的函数，而不是为结果所选的名字的函数。

我们要求不确定性的测量满足下列属性：

1. 对非负数 $p_1, \ldots, p_n$ 组成的集合，其中 $p_1+\cdots+p_n=1$，不确定性通过数 $H(p_1, \cdots, p_n)$ 给出。

2. H 应该是概率分配的一个连续函数，所以概率分配中的细小变化不会彻底改变不确定性。

3. 对于所有 $n>0, H(\frac{1}{n},\cdots,\frac{1}{n}) \leqslant H(\frac{1}{n+1},\cdots,\frac{1}{n+1})$。换句话说，在所有的输出结果都等同的情况下，不确定性就会随着可能的输出结果的增加而增长。

4. 如果 $0<q<1$，那么

$$H(p_1,\ldots,qp_j,(1-q)p_j,\ldots,p_n)=H(p_1,\ldots,p_j,\ldots,p_n)+p_jH(q,1-q)。$$

这意味着，如果第 j 个结果被分成概率分别为 qp_j 和 $(1-q)p_j$ 的两个子结果，那么熵就会增加一个值，该值等于这两个子输出之间的选择所引起的不确定性乘上这个例子开始时的概率 p_j。例如，如果我们投掷一个六面的骰子，我们可以记录两个结果：奇数和偶数，这有不确定性 $H(\frac{1}{2}, \frac{1}{2})$，现在假设我们把偶数结果分为子结果 2 和 {4，6}，那么就有 3 种可能的结果：2，{4，6} 和奇数，则有

$$H\left(\frac{1}{6},\frac{1}{3},\frac{1}{2}\right)=H\left(\frac{1}{2},\frac{1}{2}\right)+\frac{1}{2}H\left(\frac{2}{3},\frac{1}{3}\right)。$$

第一项是由偶数对奇数引起的不确定性，第二项是由偶数分裂成两个子结果而增加的不确定性。

从这些基本假设出发，香农［Shannon2］提出以下观点。

定理：令 $H(X)$ 为满足上面属性 1～4 的函数，换句话说，对每一个结果为 $\chi=\{x_1, \cdots, x_n\}$，拥有属性 $p_1, \ldots, p_n$ 的随机变量 X，函数 H 分配了一个数 $H(X)$，使之满足条件 1～4，那么 H 一定满足

$$H(p_1,\cdots,p_n)=-\lambda\sum_k p_k\log_2 p_k$$

这种形式，其中 λ 为一个正常数，而总和由满足 $p_k>0$ 的 k 决定。

通过这个定理，我们定义变量 X 的**熵（entropy）**为

$$H(X)=-\sum_{x\in\chi}p(x)\log_2 p(x),$$

熵 $H(X)$ 是对 X 的结果的不确定性的测量。

细心的读者可能注意到，当存在元素 $x\in\chi$ 且其概率为 0 时，就会出现问题。本例中我们定义 $0\log_2 0=0$，这可以通过 $x\to 0$ 时 $x\log_2 x$ 的极限来证明，以 2 为底的对数是典型的约定，在这样的例子中熵以比特为单位来测量。从现在起，我们将丢弃对数的下标，并假设正在处理的是以 2 为底的对数，X 的熵也可以解释为 $-\log_2 p(X)$ 的期望值（记为 $E[g(X)]=\sum_x g(x)p(x)$）。

下面我们看一些例子。

例：考虑抛掷一枚均匀的硬币，有两个结果，每个结果的概率均为 1/2。这个随机事件的熵为

$$-\left(\frac{1}{2}\log_2\frac{1}{2}+\frac{1}{2}\log_2\frac{1}{2}\right)=1\text{ 比特}。$$ ■

例：考虑一枚不均匀的硬币投掷 X，其正面朝上的概率为 p，反面朝上的概率为 $1-p$ $(0<p<1)$，这个事件的熵为

$$H(X)=-p\log_2 p-(1-p)\log_2(1-p)。$$

如果把 $H(X)$ 看作 p 的函数，可以知道当 $p=\frac{1}{2}$ 时熵得到最大值。 ■

例：考虑投掷一个有 n 面的骰子，有 n 个结果，每个结果的概率均为 $1/n$，熵为

$$-\frac{1}{n}\log_2(1/n)-\cdots-\frac{1}{n}\log_2(1/n)=\log_2(n)。$$ ■

熵和精确决定一个随机事件结果的是非问题的数目之间是有关系的，如果考虑抛掷一枚 $p(1)=1$ 的完全不均匀的硬币，那么 $H(X)=0$，这个结果可以解释为这个事件的值不要求任何问题来决定。如果某人抛了一个四面的骰子，那么它通过两个是非问题来获取结果，例如，数字小于 3 吗？数字是奇数吗？

通过抛两枚硬币可以得到一个更微妙的例子。令 X 表示正面朝上的次数，所以可能的结果为 {0，1，2}，概率分别为 1/4，1/2，1/4，熵为

$$-\frac{1}{4}\log_2(1/4)-\frac{1}{2}\log_2(1/2)-\frac{1}{4}\log_2(1/4)=\frac{3}{2}。$$

可知我们能够用平均 3/2 个问题来决定结果。例如，第一个问题可能是："只有一个正面朝上吗？"有一半的可能，这个问题对决定结果是足够的。另一半可能是第二个问题所需要的，例如，"是否有两个正面朝上？"所以问题的平均数等于熵。

考虑 $H(X)$ 的另一种方法是在得到 X 的结果时测量所获取信息的比特数。例如，假设 X 的结果是一个随机的 4 比特数，其中每一个可能性的概率均为 1/16，正如以前计算的，熵为 $H(X)=4$，这就是说当我们确定了 X 的值后，就收到了 4 比特的信息。

在一个类似的情形中，熵涉及到在一台计算机（二进制设备）上表示一个事件所需要的最小比特位，参考 14.3 节。记录那些结果能以 100% 确定性预测的事件是没有任何意义的，这只会造成空间的浪费。在存储信息时，我们只需将不确定的部分编码，因为那是真正信息的所在。

如果有两个随机变量 X 和 Y，它们的联合熵 $H(X,Y)$ 定义为

$$H(X,Y)=-\sum_{x\in\chi}\sum_{y\in\mathcal{Y}}p_{X,Y}(x,y)\log_2 p_{X,Y}(x,y)。$$

这正是 14.1 节中所讨论的联合随机变量 $Z=(X,Y)$ 的熵。

在一个密码体制中，我们可能想知道在已知密文的情况下密钥的不确定性，这把我们引向了**条件熵（conditional entropy）**的概念，它是指已知 X 时 Y 的不确定性程度。定义如下：

$$
\begin{aligned}
H(Y\mid X) &= \sum_x p_X(x)H(Y\mid X=x)\\
&= -\sum_x p_X(x)\Big(\sum_y p_Y(y\mid x)\log_2 p_Y(y\mid x)\Big)\\
&= -\sum_x \sum_y p_{X,Y}(x,y)\log_2 p_Y(y\mid x)。
\end{aligned}
$$

最后的等式是从关系 $p_{X,Y}(x,y)=p_Y(y\mid x)p_X(x)$ 得出的。$H(Y\mid X=x)$ 是已知 $X=x$ 时 Y 的不确定性，它是根据条件概率通过第二行圆括号中的表达式定义的。假设已知 X 的值，那么就可以通过构成这些不确定性的权重得到 Y 中总的不确定性，进而计算 $H(Y\mid X)$。

注释：前面条件熵的定义用到了 $X=x$ 情况下 Y 的熵的平均权重，其中变量 $x\in\chi$。注意 $H(Y\mid X)\neq-\sum_{X,Y}p_Y(y\mid x)\log_2(p_Y(y\mid x))$，这个和不具有信息或者不确定性应该有的那些属性。例如，如果 X 和 Y 是相互独立的，那么这个定义暗示了已知 X 时 Y 的不确定性要大于 Y 的不确定性（参考练习 15），显然不应该出现这种情况。

现在我们导出一个重要的方法——熵的链式法则，在 14.4 节将会很有用。

定理（链式法则）：$H(X,Y)=H(X)+H(Y\mid X)$。

证明：

$$
\begin{aligned}
H(X,Y) &= -\sum_{x\in\chi}\sum_{y\in\mathcal{Y}} p_{X,Y}(x,y)\log_2 p_{X,Y}(x,y)\\
&= -\sum_{x\in\chi}\sum_{y\in\mathcal{Y}} p_{X,Y}(x,y)\log_2 p_X(x)p_Y(y\mid x)\\
&= -\sum_{x\in\chi}\sum_{y\in\mathcal{Y}} p_{X,Y}(x,y)\log_2 p_X(x)-\sum_{x\in\chi}\sum_{y\in\mathcal{Y}} p_{X,Y}(x,y)\log_2 p_Y(y\mid x)\\
&= -\Big(\sum_x \log_2 p_X(x)\sum_y p_{X,Y}(x,y)\Big)+H(Y\mid X)\\
&= -\sum_x p_X(x)\log_2 p_X(x)+H(Y\mid X)\quad（因为\ \sum_y p_{X,Y}(x,y)=p_X(x)）\\
&= H(X)+H(Y\mid X)。
\end{aligned}
$$

□

链式法则告诉了我们些什么呢？它认为联合事件（X，Y）的不确定性等于事件 X 的不确定性加上事件 X 发生的情况下事件 Y 的不确定性。

现在我们陈述一下关于熵的另外 3 个结论。

定理：

1. $H(X)\leqslant\log_2|\chi|$，$|\chi|$ 表示 χ 中的元素个数，当且仅当 χ 中的元素都相等的情况下等式才成立。

2. $H(X,Y)\leqslant H(X)+H(Y)$。

3. （条件作用减少熵）$H(Y\mid X)\leqslant H(Y)$，当且仅当 X 与 Y 相互独立时等式成立。

第一个结论说明了当概率分配均衡时是最不确定的，返回到抛掷不均匀硬币的例子，当 $p=\frac{1}{2}$ 时熵达到最大值，这可以扩展到有更多可能结果的事件。(1) 的证明可以参考 [Welsh，p.5]。

第二个结论说明包含在（X，Y）中的信息至多等于包含在 X 中的信息加上包含在 Y 中

的信息，不相等的原因很可能在于 X 和 Y 提供的信息是互相交迭的（当 X 和 Y 不相互独立时的情况）。对（2）的证明，参考［Stinson］。

第三个结论是信息论中最重要的结论之一，它的解释是非常简单的，它说明在 X 事件发生的前提下随机事件 Y 的不确定性要小于单独事件 Y 的不确定性，也就是说，X 仅仅能够告诉你关于事件 Y 的信息，而不能知道对 Y 任何更多的不确定性。

第三个结论是第二个结论和链式法则的简单推论：

$$H(X)+H(Y\mid X)=H(X,Y)\leqslant H(X)+H(Y)。$$

14.3 哈夫曼编码

信息论起源于 20 世纪 40 年代克劳德·香农的产生巨大影响的论文，香农精确的信息理论背后的主要动机之一是找到一种表示数据的更加紧凑的方法，简而言之，他参与了压缩问题的研究。本节我们将简要地谈谈熵和压缩之间的关系，并介绍一种更加简洁地表示数据的方法——哈夫曼编码。

要想了解更多关于怎样压缩数据的信息，请参考［Cover-Thomas］或者［Nelson-Gailly］。

例如：假设有一个含有 a，b，c，d 四个字母的字母表，这些字母以如下所示的频率出现在文档中。

a	b	c	d
.5	.3	.1	.1

我们可以用二进制串 00 表示 a，01 表示 b，10 表示 c，11 表示 d，这意味着消息中每个字母平均位数为 2 比特。然而，假设我们用 1 表示 a，01 表示 b，001 表示 c，000 表示 d，那么每个字母的平均位数为

$$(1)(.5)+(2)(.3)+(3)(.1)+(3)(.1)=1.7$$

（a 的位数乘以 a 的频率，加上 b 的位数乘以 b 的概率，等等），可见字母的编码是更加有效的。 ■

一般情况下，有一个结果在集合 χ 中的随机变量，我们想通过一种有效的方法用二进制来表示这个结果，也就是说，每个结果位数的平均数应该尽可能地小。

这样一个过程的早期的例子是莫尔斯编码，它用点和横的序列来表示字母，并且已经运用在用电报传送信息上。莫尔斯询问印刷工哪一个字母用得最多，使较常用的字母有较短的表示符，例如：e 被表示成 ·，t 被表示成 －，但是 x 被表示成 －··－，z 被表示成 －－··。

哈夫曼提出了一种新方法，主要内容是列出所有的结果和它们的概率，最小的两个被分配为 1 和 0，然后联合起来形成一个有较大概率的结果。同样的过程被应用到这个新的列表，分配 1 和 0 给列表中最小的两个，然后联合它们形成新的列表，持续这个过程直到只剩下一个结果，经过这个过程向后读就可以得到一个二进制串，记录被分配给已知结果和包含该结果的联合的这些位。最好通过一个例子来解释。

如前例所示，假设有输出 a，b，c，d 的概率分别为 .5，.3，.1，.1，图 14.1 展示了这

个过程。

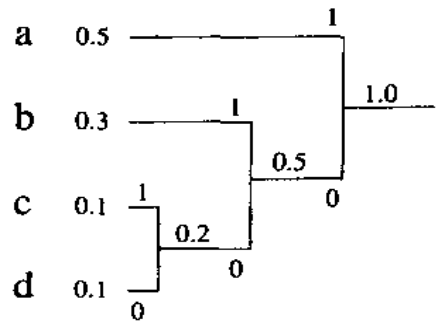


图 14.1　一个哈夫曼编码的例子

既然要选择两个最低的输出，那么我们随机分配哪一个为 0，哪一个为 1，通过这个图表向后搜索，可看到 a 仅仅收到一个 1，b 收到 01，c 收到 001，d 收到 000，这些与前面为每个字母分配较少位数是一样的。

哈夫曼编码的一个有用的特点是可以一次一个字母地读取信息。例如：串 011000 仅仅能够读为 bad；甚至，只要我们一读前两位 01，就知道第一个字母是 b。

假设改为以相反的顺序来书写分配给字母的位，则 b 为 10，而 c 为 001，那么信息 101000 直到所有的位都被读完后才能确定，因为它可能以 bb 或 ba 开头。

更糟糕的是，假设我们分配 0 给 a 代替 1，那么信息 aaa 和 d 将是一样的。很容易说明哈夫曼编码避免了这两个问题。

每个输出的平均位数与熵是密切相关的。

定理：令 L 表示哈夫曼编码中随机变量 X 的每个输出的平均位数，那么

$$H(X) \leqslant L < H(X) + 1。$$

这个结论与熵测量了 X 的输出中包含了信息的多少位的解释是一致的。我们在这儿忽略了定理的证明。在本例中，熵

$$H(X) = -(.5\log_2(.5) + .3\log_2(.3) + .1\log_2(.1) + .1\log_2(.1)) \approx 1.685。$$

14.4　完全保密

直观上，一次一密的加密体制提供了完全保密性，熵允许我们用精确的术语来描述它。假设有一个可能的明文为 p，密文为 C、密钥为 k 的密码体制，p 中的每一个明文都有一个确定的发生概率，并且某些比另外的可能性更大，k 中密钥的选择总是假定不受明文选择的约束，C 中可能的密文依赖于 p 和 k 的概率，从而有各种不同的概率。

如果伊芙截取了一个密文，她能得到多少关于密钥的信息呢？换句话说，$H(K|C)$ 为多少呢？最初，密钥的不确定性为 $H(K)$，那么密文的获取降低不确定性了吗？

例：假设有 3 个可能的明文：a，b，c，概率分别为 .5，.3，.2，两个密钥 k_1，k_2 的概率分别为 .5，.5，可能的密文为 U，V，W。令 e_k 表示密钥为 k 的加密函数，假设

$$e_{k_1}(a) = U, e_{k_1}(b) = V, e_{k_1}(c) = W$$

$$e_{k_2}(a) = U, e_{k_2}(b) = W, e_{k_2}(c) = V。$$

令 $p_P(a)$ 表示明文为 a 的概率，依此类推。密文是 U 的概率为

$$p_C(U) = p_K(k_1)p_P(a) + p_K(k_2)p_P(a)$$
$$= (.5)(.5) + (.5)(.5) = .50。$$

类似地，计算 $p_C(V) = .25, p_C(W) = .25$。

假设某人截取一段密文，这可以提供一些明文的信息。例如，如果密文为 U，那么马上可以推出明文为 a；如果密文为 V，那么明文为 b 或者 c。

甚至我们可以知道更多：密文为 V 的概率是 .25，所以在密文为 V 的前提下明文为 b 的条件概率是

$$p(b \mid V) = \frac{p_{(P,C)}(b,V)}{p_C(V)} = \frac{p_{(P,K)}(b,k_1)}{p_C(V)} = \frac{(.3)(.5)}{.25} = .6。$$

相似地，$p(c \mid V) = .4$，$p(a \mid V) = 0$，我们也可以计算出

$$p(a \mid W) = 0,\ p(b \mid W) = .6,\ p(c \mid W) = .4。$$

注意到明文的初始概率为 .5，.3 和 .2；密文的获取允许我们去修改这些概率。因此，密文提供了明文的信息，通过条件熵的概念可以把它量化。首先，明文的熵为

$$H(P) = -(.5\log_2(.5) + .3\log_2(.3) + .2\log_2(.2)) = 1.485。$$

已知 C 时 P 的条件熵为

$$H(P \mid C) = -\sum_{x \in \{a,b,c\}} \sum_{Y \in \{U,V,W\}} p(Y)p(x \mid Y)\log_2(p(x \mid Y)) = .485。$$

因此，在当前的例子中，当密文被获取时，明文的不确定性就会降低。 ■

另一方面，我们怀疑对于一次一密加密体制来说，密文不生成任何以前不了解的明文的信息。换句话说，明文的不确定性与已知密文时它的不确定性是相同的。这也引出了以下的定义和定理。

定义：如果 $H(P \mid C) = H(P)$，那么一个密码体制有**完全的保密性**（**perfect secrecy**）。

定理：一次一密加密体制有完全的保密性。

证明。记得基本的设置如下：有一个含有 Z（例如，Z 可能是 2 或者 26）个字母的字母表，可能的明文是由长度为 L 的字符串组成的，密文是长度为 L 的字符串，有 Z^L 个密钥用来表示各种各样被采用的变化，每个密钥都是由长度为 L 的序列组成的，密钥被随机地选择，所以每一个发生的概率为 $1/Z^L$。

令 $c \in C$ 表示一个可能的密文。如以前一样，计算 c 发生的概率：

$$p_C(c) = \sum_{\substack{x \in P, k \in K \\ e_k(x) = c}} p_P(x)p_K(k)。$$

这里 $e_k(x)$ 表示用密钥 k 加密 x 所获得的密文，这个和包括所有那些满足用 k 加密 x 形成 c 的 x，k 组合。注意，我们已经根据 P 和 K 的独立性把联合概率 $p_{(P,K)}(x, k)$ 写成单独概率的乘积。

在一次一密加密体制中，每一个密钥有相同的概率 $1/Z^L$，所以在上面的和式中可以用 $1/Z^L$ 代替 $p_K(k)$，得到

$$p_C(c) = \frac{1}{Z^L} \sum_{\substack{x \in P, k \in K \\ e_k(x) = c}} p_P(x)。$$

现在运用一次一密加密体制的另一个重要特性：对每一个明文 x 和密文 c，确切地说只有一个密钥 k 满足 $e_k(x)=c$，因此，一旦前述的和式中每一个 $x\in p$ 确实发生时，就可以得到 $Z^{-L}\sum_{x\in p}p_p(x)$，但是因为所有可能的明文的概率之和为 1，所以得到

$$p_C(c)=\frac{1}{Z^L}。$$

这确认了我们所怀疑的：每一个密文以相同的概率发生。

现在计算一些熵。因为 K 和 C 对于所有 Z^L 个可能性都有相同的概率，所以可得到

$$H(K)=H(C)=\log_2(Z^L)。$$

现在用两种不同的方法计算 $H(P,K,C)$，因为知道 (P,K,C) 与知道 (P,K) 是一样的，所以有

$$H(P,K,C)=H(P,K)=H(P)+H(K)。$$

最后一个等式是由于 P 和 K 是相互独立的，对于一次一密加密体制来说 C 和 P 决定 K，所以知道 (P,K,C) 与知道 (P,K) 是一样的，因此，

$$H(P,K,C)=H(P,C)=H(P\mid C)+H(C)。$$

最后一项等式是链式法则，可合并这两个表达式，又因为 $H(K)=H(C)$,所以得到 $H(P\mid C)=H(P)$。这证明了一次一密加密体制有完全的保密性。 □

上面的证明产生了下列更多的一般性结论。令#k 表示可能的密钥数目，依此类推。

定理：考虑一个密码体制，如果满足下列条件：

1. 每个密钥有概率 1/#k。
2. 对于每一个 $x\in p$，$c\in C$，确实有一个 $k\in k$ 满足 $e_k(x)=c$。

那么这个密码体制有完全的保密性。

很容易从条件（2）推论出#C=#k。反过来，可以证明如果#p=#C=#k，并且体制有完全的保密性，那么条件（1）和（2）成立（参考 [Stinson，定理 2.4]）。

自然我们会问到前面的概念怎样应用到 RSA 上，可能令人吃惊的回答是 $H(P\mid C)=0$，也就是说,密文决定了明文,原因是熵没有考虑计算时间,可能花费几十亿年的时间来因数分解 n 这个事实是不相关的,相关的是所有需要用来重新获取包含在已知的 n,e 和 c 中的明文的信息。

对 RSA 来说更相关的概念是破解体制的计算复杂性。

14.5 英文的熵

在一篇英文文档中，每个字母有多少信息能被获得呢？如果有一个随机的字母序列，其中每个字母都以 1/26 的概率出现，那么熵将是 $\log_2(26)=4.70$；所以每个字母包含 4.7 位信息。如果包括空格，得到 $\log_2(27)=4.75$。但是这些字母的概率是不相等的：a 的概率为 .082，b 的概率为 .015，等等（参考 2.3 节），因此，得到

$$-(.082\log_2 .082+.015\log_2 .015+\cdots)=4.18。$$

虽然这样，这并不能断定整个过程。假设有一个正在研究的字母序列，最后一个字母是什么有非常小的不确定性；很容易猜到它是g 。相似地，如果看到字母 q，下一个字母很可能为 u，

因此，现有的字母经常给出下一个字母的信息，这意味着那个字母并没有携带更多另外的信息，这说明前面计算的熵仍然太高了。如果用 $26^2=676$ 个连字（一个连字是两个字母的联合）的频率表，可以计算在已知前一个字母的前提下该字母的条件熵为 3.56。用三连字频率表，我们发现在已知一个字母的前两个字母的前提下该字母的条件熵接近于 3.3。一般地说，这表明如果知道一篇文档中两个连续的字母，那么下一个字母携带 3.3 位额外的信息。因此，如果有一篇长文档，我们应该预料到至少能够以大约 3.3/4.7 = .7 的压缩率压缩它。

令 L 表示英文字母，L^N 表示 N 个字母的联合，定义英文的熵为

$$H_{\text{English}} = \lim_{N\to\infty}\frac{H(L^N)}{N},$$

其中 $H(L^N)$ 表示 N 字母联合的熵。这给出了长文档中每个字母的平均信息量，如果关于这个文档我们已经了解了很多，那么它也表示猜测下一个字母的平均不确定性。如果这些字母都是相互独立的，那么连字 qu 的概率等于 q 的概率乘以 u 的概率，于是有 $H(L^N)=N\cdot H(L)$，它的极限为 $H(L)$，$H(L)$ 是一个字母频率的熵，但是如同在连字和三连字频率中看到的那样，字母的交互作用降低了 $H(L^N)$ 的值。

我们怎样计算 $H(L^N)$ 呢？计算 100 连字频率是不可能的，甚至列出它们的最大共性或得到一个近似值也是很困难的，香农提出了下面的方法。

假设有一台理想的预报器，在某种意义上，已知文档中的一个长字符串，可以计算下一个将出现的字母的概率，然后以最大的可能性猜测这个字母。如果正确，记录这个字母，并写一个 1，如果不正确，猜测第二大可能的字母，如果正确，写一个 2，依此类推，通过这种方法，得到一个数字序列。例如，考虑文本 *itissunnytoday*，假设预报器认为 t 最有可能是第一个字母，结果是错的；第二次猜测是 i，这是正确的，所以我们写 i，并在 i 下写个 2，然后预报器预测 t 是下一个字母，这是正确的，在 t 下写个 1。继续下去，假设第一次猜测就找到了 i，依此类推，得到下面的这种情况：

i	*t*	*i*	*s*	*s*	*u*	*n*	*n*	*y*	*t*	*o*	*d*	*a*	*y*
2	1	1	1	4	3	2	1	4	1	1	1	1	1

利用预报器可以重新建立这个文本，预报器认为它对第一个字母的第二次猜测是 i，所以我们知道第一个字母是 i，预报器认为对下一个字母的第一次猜测是 t，所以我们知道 t 是下一个，再下一个的第一次猜测为 i，依此类推。

这意味着如果有一台预测的机器，那么由于我们能够重建这个文档，所以可以不丢失任何信息地把文本转变成数字串。当然，可以尝试写一个计算机程序来进行预测，但是香农认为最好的预报器是一个讲英语的人。当然，人不像机器那样具有确定性，重复这个实验（假设这个人从一开始就忘了这个文本）可能不生成一个同样的结果，所以重建这个文本可能有一些困难，但是把一个人近似成一台理想的预报器仍然是一个合理的假设。

已知与一个文本相对应的整数序列，可以计算每一个数字的频率，令

$$q_i = \text{数字 } i \text{ 的频率},$$

因为文本和数字序列可以被互相重建，所以它们的熵一定是相同的。当这些数字相互独立时，数字序列的熵取得最大值。本例中，熵为 $-\sum_{i=1}^{26}q_i\log_2(q_i)$。然而，这些数字一般不是相互独立的。例如，如果有一对连续的 1，那么预报器可能猜测这个单词剩下的部分将有更多的 1，即便如此，可以得到熵的上界，这通常比通过字母的频率得到的上界要好得多。此

外，香农也找到了熵的下界，他的结论是

$$\sum_{i=1}^{26} i(q_i - q_{i+1})\log_2(i) \leqslant H_{\text{English}} \leqslant -\sum_{i=1}^{26} q_i\log_2(q_i)。$$

事实上，由于有实验误差并且我们真正考虑的是 N 趋向无穷大时的极限，所以这些仅仅是近似的上界和下界。

这些结论允许对英文熵的实验估计。艾丽斯选择一个文本，鲍勃预测第一个字母，持续下去直到做出正确的预测，艾丽斯记录预测出的数字，然后鲍勃尽量去预测第二个字母，再一次记录预测出的数字，鲍勃用这种方法继续预测每个字母，当他正确时，艾丽斯告诉他并记录预测出的数字。作为一个典型的实验结论，香农给出了表 14.1，注意包括空格，但忽略了标点符号，所以有 27 种可能性：有 102 个符号，其中 79 个 1，8 个 2，3 个 3，等等。可得到

$$q_1 = 79/102,\ q_2 = 8/102,\ q_3 = 3/102,\ q_4 = q_5 = 2/102,$$
$$q_6 = 3/102, q_7 = q_8 = q_{11} = q_{15} = q_{17} = 1/102。$$

因此熵的上界为

$$-\left(\frac{79}{102}\log_2\frac{79}{102} + \cdots + \frac{1}{102}\log_2\frac{1}{102}\right) \approx 1.42。$$

注意，既然 $0\log_2 0 = 0$，那么包含 $q_i = 0$ 的这些项就可以被忽略，下界为

$$1\cdot\left(\frac{79}{102} - \frac{8}{102}\right)\log_2(1) + 2\cdot\left(\frac{8}{102} - \frac{3}{102}\right)\log_2(2) + \cdots + \approx .72。$$

因此一个合理的估计是英文的熵近似为 1，也可能比 1 稍微大一些。

如果想要传送一个长的英文文本，我们可以用一个 5 位的字符串来表示每个字母（和空格），这样像前面提到的长度为 102 的文本将需要 510 位。如果这些字母是独立且相等的，那么用一些类似的方法就是必要的。然而，假设对表 14.1 中的信息 1，1，1，5，1，1，2，... 进行哈夫曼编码。令

$$1\leftrightarrow 1 \quad 2\leftrightarrow 110 \quad 3\leftrightarrow 1010 \quad 4\leftrightarrow 0100$$
$$5\leftrightarrow 11100 \quad 6\leftrightarrow 0010 \quad 7\leftrightarrow 01100 \quad 8\leftrightarrow 11000$$
$$11\leftrightarrow 01000 \quad 15\leftrightarrow 10000 \quad 17\leftrightarrow 100000。$$

直到 27 的所有数字都能被 6 位或更多比特位的联合来表示。传送这条信息要求

$$79\cdot 1 + 8\cdot 3 + 3\cdot 4 + 2\cdot 4 + \cdots + 1\cdot 6 = 171\ \text{位},$$

每个字母 1.68 位。

注意，每个字母有 5 位仅仅比随机的熵 4.75 大一点，每个字母 1.68 位比我们对英文熵的估计稍微大一点，这与至多为 1 的熵不同于哈夫曼编码的平均长度的结论是一致的。

考虑前面的熵计算的一种方法是认为英文大约有 75% 的冗余。也就是说，如果传送一条用标准写作英文书写的长信息，与最佳压缩文本比较，比率近似为 4:1（也就是说，随机熵 4.75 除以英文熵后约为 1）。在我们的例子中，得到一个约为 3:1 的比率（也就是 4.75/1.68）。

定义英文的**冗余（redundancy）**为

$$R = 1 - \frac{H_{\text{English}}}{\log_2(26)}。$$

那么 R 近似为 0.75，也就是前面提到的 75% 的冗余。

表 14.1 香农关于英文熵的实验

t	h	e	r	e		i	s		n	o		r	e	v	e	r	s	e	
1	1	1	5	1	1	2	1	1	2	1	1	15	1	17	1	1	1	2	1

o	n		a		m	o	t	o	r	c	y	c	l	e		a	
3	2	1	2	2	7	1	1	1	1	4	1	1	1	1	1	3	1

f	r	i	e	n	d		o	f		m	i	n	e		f	o	u	n	d	
8	6	1	3	1	1	1	1	1	1	1	1	1	1	1	6	2	1	1	1	1

t	h	i	s		o	u	t		r	a	t	h	e	r	
1	1	2	1	1	1	1	1	1	4	1	1	1	1	1	1

d	r	a	m	a	t	i	c	a	l	l	y		t	h	e	
11	5	1	1	1	1	1	1	1	1	1	1	6	1	1	1	1

o	t	h	e	r		d	a	y	
1	1	1	1	1	1	1	1	1	1

单一距离（Unicity Ditance）

假设有一个密文，有多少密钥可将它解密成一些有意义的东西呢？如果文本足够长，我们怀疑只有惟一的密钥和惟一相对应的明文。一个密码体制的单一距离 n_0 是指预计只有惟一有意义的明文所对应的密文的长度，单一距离的大体估计值为

$$n_0 = \frac{\log_2 |K|}{R\log_2 |L|},$$

其中 $|K|$ 是可能的密钥数目，$|L|$ 是字母或符号的数目，R 表示冗余（参考 [Stinson]）。我们取 $R = .75$（至于我们的语言中是否包含空格，区别很小）。

例如，考虑有 26! 个密钥的替换密码。可得

$$n_0 = \frac{\log_2 26!}{.75\log_2 26} \approx 25.1。$$

这意味着如果一个密文长度为 25 或更长，那么我们预计通常仅仅有一个有意义的明文，当然，如果有一个长度为 25 的密文，它们很可能有几个字母没有出现，因此，有几个可能的密钥，它们都能将这个密文解密为相同的明文。

另外一个例子，考虑仿射密码，它有 312 个密钥，所以

$$n_0 = \frac{\log_2 312}{.75\log_2 26} \approx 2.35。$$

这仅是一个非常粗略的近似值。显然，得到惟一的译码要使用更多的字母，但是估计值 2.35 说明，在大多数情况下对于仿射密码来讲产生惟一的译码用非常少的字母就足够了。

最后，考虑长度为 N 的信息的一次一密加密方式，加密是对每个字母单独地模 26 变换，所以有 26^N 个密钥，可得到估计值

$$n_0 \approx \frac{\log_2 26^N}{.75\log_2 26} \approx 1.33N。$$

在本例中，为了得到一个惟一的译码，需要比全部密文更多的字母，这也反映了所有明文可能适合于任何密文的事实。

14.6 习 题

1. 令 X_1 和 X_2 表示两个相互独立的均匀硬币投掷，得到熵 $H(X_1)$和联合熵 $H(X_1,X_2)$，为什么 $H(X_1,X_2) = H(X_1) + H(X_2)$ 呢？

2. 考虑一个不均匀的硬币，会产生两个结果：正面朝上和反面朝上，分别有概率 p（正面朝上）$=p$ 和 p（反面朝上）$=1-p$。

(a) 如果硬币被抛两次，结合它们各自的概率，可能的结果会是怎样的？

(b) 证明 (a) 中的熵为 $-2p\log_2(p) - 2(1-p)\log_2(1-p)$。如果不计算 (a) 中的概率，如何来预测这个值？

3. 一个随机变量 X 分别以概率 $\frac{1}{2}$，$\frac{1}{2^2}$，…，$\frac{1}{2^n}$，…表示值 1，2，…，n，…，计算熵 $H(X)$。

4. 令 X 表示取整数值的随机变量，X 在范围 $[0, 2^8-1]$ 中所有值的概率都是相同的，为 1/2，X 在范围 $[2^8, 2^{32}-1]$ 中所有值的概率都是相同的，为 1/2，计算 $H(X)$。

5. 令 X 表示取值 −2，−1，0，1，2 的随机事件，所有值都有肯定的概率。在 Y 如下所示时，$H(X)$ 和 $H(Y)$ 之间不等式/等式的关系是怎样的？

(a) $Y = 2^X$

(b) $Y = X^2$

6. (a) 本题我们探讨一下随机变量 X 的熵和随机变量函数 $f(X)$ 的熵之间的关系。下面是对 $H(f(X)) \leqslant H(X)$ 的简单证明，解释一下每一步使用了什么原理。

$$H(X, f(X)) = H(X) + H(f(X) \mid X) = H(X),$$

$$H(X, f(X)) = H(f(X)) + H(X \mid f(X)) \geqslant H(f(X))。$$

(b) 令 X 取值为 ± 1，$f(x) = x^2$，证明 $H(f(X)) < H(X)$ 是可能成立的。

(c) 在 (a) 中，证明当且仅当 f 为一对一函数（更精确地讲，f 在没有非零概率的 X 的输出集合上是一对一的）时等式成立。

(d) 前面的结论能够用来研究序列的游程编码的过程。游程编码是通常应用于数据压缩的一门技术。假设 X_1，X_2，…，X_n 是值为 0 或 1 的随机变量，这个随机变量的序列能够被看作一个二进制源的输出表示，X_1，X_2，…，X_n 的游程编码是一个序列 $\mathbf{L} = (L_1, L_2, \cdots, L_k)$，该序列用来表示拥有相同值的连续符号的长度。例如，序列 110000100111 有一个游程序列 $\mathbf{L} = (2, 4, 1, 2, 3,)$，观察到 $\mathbf{L}$ 是 X_1，X_2，…，X_n 的函数，说明 $\mathbf{L}$ 和 X_1 惟一确定 X_1，X_2，…，X_n。$\mathbf{L}$ 和 X_n 确定 X_1，X_2，…，X_n 吗？运用这些观察结果和以前的结论，比较 $H(X_1, X_2, \cdots, X_n)$，$H(\mathbf{L})$ 和 $H(\mathbf{L}, X_i)$。

7. 一个包里有 5 个红色球、3 个白色球和 2 个黑色球，除了颜色之外其他方面都是完全一样的。

(a) 放回地选择两个球，这个实验的熵为多少？

(b) 不放回地选择两个球的熵为多少？

(注意：在这两问中球的顺序问题；也就是说，先红后白与先白后红是不同的。)

8. 我们经常遇上 n 个随机事件序列的情况，例如，一个文本是一个长的字母序列，我

们只关心当 n 增加时联合熵的增长率，定义随机事件序列 $\mathbf{X}=\{X_k\}$ 的熵比率为

$$H_{\infty}(\mathbf{X})=\lim_{n\to\infty}\frac{1}{n}H(X_1,X_2,\cdots X_n)。$$

（a）针对某种语言的一个非常粗糙的模型是假设一篇文本中的并发字母都是独立的，并且有同样的概率分布，使用这个条件证明熵比率等于 $H(X_1)$。

（b）一般来讲，随机变量中有依赖性，假设 X_1，X_2，$\cdots X_n$ 有相同的概率分布，但它们在某种程度上是相互依赖的（例如，如果给你字母 TH，你肯定猜测下一个字母为 E），证明

$$H(X_1,X_2,\cdots X_n)\leqslant\sum_k H(X_k)$$

因而

$$H_{\infty}(\mathbf{X})\leqslant H(X_1)$$

（假设极限 H_{∞} 存在）。

9. 假设有一个仅仅有两种可能明文的密码体制，明文 a 以 1/3 的概率发生，b 以 2/3 的概率发生。有两个密钥 k_1 和 k_2，每一个被使用的概率为 1/2，密钥 k_1 加密 a 为 A，加密 b 为 B。密钥 k_2 加密 a 为 B，加密 b 为 A。

（a）计算明文的熵 $H(P)$。

（b）计算已知密文的前提下明文的条件熵 $H(P|C)$。（提示：可以不经过额外的计算，通过把这个体制与另一个著名的体制相匹配来处理。）

10. 考虑密码体制 $\{P, K, C\}$。

（a）解释为什么 $H(P,K)=H(C,P,K)=H(P)+H(K)$。

（b）假设体制有完全保密性，证明 $H(C,P)=H(C)+H(P)$ 和 $H(C)=H(K)-H(K|C,P)$。

（c）假设体制有完全保密性，并且对每组明文和密文至多有一个相应的密钥来进行编码。证明 $H(C)=H(K)$。

11. 证明对于密码体制 $\{P,K,C\}$，有

$$H(C|P)=H(P,K,C)-H(P)-H(K|C,P)=H(K)-H(K|C,P)。$$

12. 考虑 Shamir 秘密共享方案，遵守规则：20 个人的集合中任何 5 个可以决定秘密 K，少于 5 个就不能决定。令 $H(K)$ 表示 K 的选择的熵，$H(K|S_1)$ 表示在已知提供给第一个人的信息的前提下 K 的条件熵。$H(K)$ 和 $H(K|S_1)$ 的相对大小如何（大于、小于还是相等）?

13. 令 X 表示值为 1，2，3，…，36 的随机事件，所有值都有相同的概率。

（a）熵 $H(X)$ 为多少?

（b）令 $Y=X^{36}\pmod{37}$，$H(Y)$ 为多少?

14. 对于任意的 $1\leqslant i\leqslant n$，令 $p_i\geqslant 0$。证明当 $p_1=\cdots=p_n$ 时，$-\sum_i p_i\log_2 p_i$ 产生最大值，其中满足限制条件 $\sum_i p_i=1$。（提示：在这个问题中，拉格朗日乘数可能是有用的。）

15. （a）假设我们定义 $\tilde{H}(Y|X)=-\sum_{x,y}p_Y(y|x)\log_2 p_Y(y|x)$。证明：如果 X 和 Y 是相互独立的，并且 X 有 $|\chi|$ 个可能的输出，那么 $\tilde{H}(Y|X)=|\chi|H(Y)\geqslant H(Y)$。

（b）用（a）证明 $\tilde{H}(Y|X)$ 不是已知 X 的前提下 Y 的不确定性的一个好的描述。

第 15 章

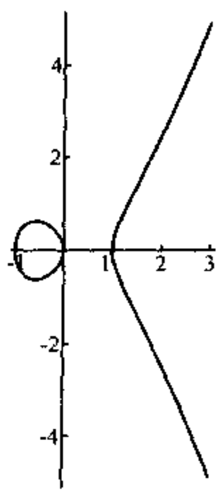
椭圆曲线

在 20 世纪 80 年代中期，Miller 和 Koblitz 将椭圆曲线引进了密码学中。Lenstra 说明了怎样用椭圆曲线来因数分解整数。从那时起，椭圆曲线已经在许多涉及密码学的地方起着日益重要的作用。它的一个优点是：与采用更大密钥尺寸的常规密码体制相比，它似乎提供了一个安全级。例如，据 Blake 等人估计，某些有 4096 位密钥的传统体制能够被 313 位的椭圆曲线体制替代，在硬件实现中用很短的数字就可以表示一个相当可观的存储。

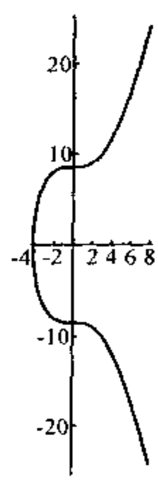
本章我们将介绍其中最重要的部分。更多关于椭圆曲线和它们的密码应用的细节，请参考 [Black et al.]。

15.1 加法定律

椭圆曲线 E 是方程 $E: y^2 = x^3 + ax + b$ 的图形，其中 a, b 在一个适当的集合中（有理数、复数、模 n 整数，等等）。也包括一个表示为 ∞ 的“**趋向于无穷大的点**”（**point at infinity**），这个点很容易被当作 y 轴的顶点，在投影几何学的上下文中它被严格地论述，但是这个直觉的符号已经满足了我们的需要，y 轴的底和顶被视为一体，所以 ∞ 也位于 y 轴的底部。当我们在实数范围内工作时，图形 E 有两种可能的形式，取决于这个关于 x 的三次多项式有一个实根还是三个实根。例如，$y^2 = x(x+1)(x-1)$ 和 $y^2 = x^3 + 73$ 的图形如下所示：



$y^2 = x(x+1)(x-1)$



$y^2 = x^3 + 73$

假设三次多项式 x^3+ax+b 没有重根，这意味着我们把诸如 $y^2=x^2(x-1)$ 这样的图形排除在外，这样的曲线将在 15.3 节进行讨论。

历史点（Historical point）：椭圆曲线不是椭圆，它们是从与

$$\int_{z_1}^{z_2}\frac{dx}{\sqrt{x^3+ax+b}} \text{和} \int_{z_1}^{z_2}\frac{xdx}{\sqrt{x^3+ax+b}}$$

这样的椭圆积分（elliptic integrals）的关系中得到所说的名字的，这些椭圆积分出现在椭圆弧长的计算中。

技术点（Technical point）：大多数情况下，形如 $y^2=x^3+ax+b$ 这样的方程对于椭圆曲线是足够的。虽然这样，有时考虑形式为

$$y^2+a_1xy+a_3y=x^3+a_2x^2+a_4x+a_6$$

的方程给出的椭圆曲线也是很有必要的，其中 a_1，...，a_6 是常量。如果我们正在进行模 p 运算，其中 $p>3$ 是素数，或者我们正在实数、有理数或复数范围内进行运算，那么变量的简单变换就把目前的方程转换成 $y^2=x^3+ax+b$ 的形式。然而，如果我们正在进行模 2 或模 3 运算，或者在特征值为 2 或 3 的典型有限域（也就是，$1+1=0$ 或 $1+1+1=0$）内进行运算，那么需要采用更普通的形式，在特征值为 2 的典型域内的椭圆曲线将在 15.4 节中简要提到。

已知 E 上的两点 P_1 和 P_2，如下所示（图 15.1）我们能得到第三个点 P_3：通过 P_1 和 P_2 两点画直线 L（如果 $P_1=P_2$，那么画 E 在 P_1 点的切线）。直线 L 与 E 在第三个点 Q 相交，通过 x 轴（即改变 y 成 $-y$）反射 Q 点，得到点 P_3。通过式 $P_1+P_2=P_3$，在 E 上定义一个加法定律。注意，这与在平面上增加一个点是不同的。

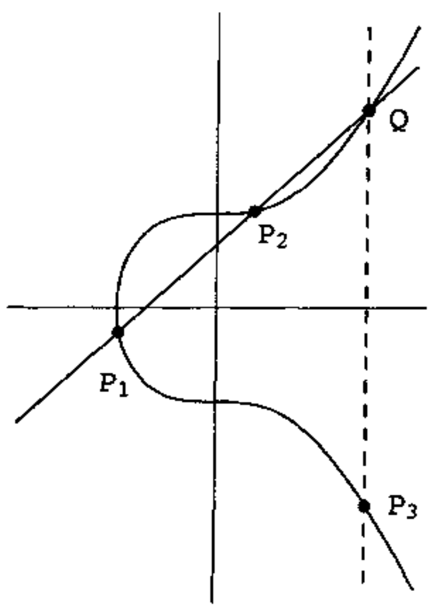


图 15.1　在椭圆曲线上增加点

例：假设 E 是由方程 $y^2=x^3+73$ 定义的，令 $P_1=(2,9)$，$P_2=(3,10)$，通过 P_1 和 P_2 的直线 L 为 $y=x+7$，将它代入 E 的方程中得到 $(x+7)^2=x^3+73$，即 $x^3-x^2-14x+24=0$，因为 L 在 P_1 和 P_2 点与 E 相交，所以我们已经知道了两个根，也就是 $x=2$ 和 $x=3$。另外，三

个根的和等于 x^2 的系数的相反数（练习 1），结果为 1。于是第三个相交的点有 $x=-4$，既然 $y=x+7$，得到 $y=3$，$Q=(-4,3)$，因此，$P_3=(-4,-3)$。

现在假设我们想把点 P_3 加到它自身，E 在 P_3 点的切线的斜率可以通过对 E 的方程进行隐函数微分得到：

$$2ydy=3x^2dx, \text{所以} \frac{dy}{dx}=\frac{3x^2}{2y}=-8,$$

其中我们把 P_3 的值 $(x,y)=(-4,-3)$ 代入进去。本例中，直线 L 为 $y=-8(x+4)-3$，代入 E 的方程中得到 $(-8(x+4)-3)^2=x^3+73$，因此 $x^3-(-8)^2x^2+\ldots=0$，三个根的和为 64（等于 x^2 系数的相反数），因为直线 L 是 E 的切线，所以认为 $x=-4$ 是一个二重根，第三个根为 $64-2(-4)=72$，y（用 L 的方程）的相应值为 -611，把 y 变成 $-y$，得到

$$P_3+P_3=(72,611)。$$ ■

如果我们尽量去计算 $P+\infty$ 会发生什么呢？通过 ∞ 和 P 点的直线是垂直的，它在 $P=(x,y)$ 和 $(x,-y)$ 处与 E 相交，当我们经由 x 轴反射 $(x,-y)$ 时，又得到了 $P=(x,y)$。因此，

$$P+\infty=P。$$

同样我们也能删除点。首先，观察到通过 (x,y) 和 $(x,-y)$ 的线是垂直的，所以与 E 相交的第三个点是 ∞，经由 x 轴反射后点仍然是 ∞（这就是我们所说的 ∞ 位于 y 轴的顶部和底部的意思），因此，

$$(x,y)+(x,-y)=\infty。$$

既然 ∞ 扮演着附加的恒等式（就像加法里的 0）的角色，所以可以定义

$$-(x,y)=(x,-y),$$

减去点 $P-Q$，实际上只是简单地增加点 P 和 $-Q$。

表示这个加法定律的另一种方法是

$$P+Q+R=\infty \Leftrightarrow P,Q,R \text{ 在同一直线上。}$$

（参考练习 7）。

为了计算，我们可以忽略几何解释，仅仅根据如下的公式工作。

加法定律：令 E 由 $y^2=x^3+ax+b$ 给出，

令 $$P_1=(x_1,y_2),P_2=(x_2,y_2)。$$

那么

$$P_1+P_2=P_3=(x_3,y_3),$$

其中，

$$x_3=m^2-x_1-x_2$$

$$y_3=m(x_1-x_3)-y_1$$

$$\text{且 } m=\begin{cases}(y_2-y_1)/(x_2-x_1) & \text{如果 } P_1\neq P_2\\ (3x_1^2+a)/(2y_1) & \text{如果 } P_1=P_2。\end{cases}$$

如果斜率 m 是无穷大的，那么 $P_3=\infty$。对所有的点 P 存在一个加法定律：$\infty+P=P$。

可以证明加法定律满足结合律：

$$(P+Q)+R=P+(Q+R)。$$

也满足交换律：

$$P+Q=Q+P。$$

因此，当增加几个点的时候，这些点以什么顺序被添加和它们怎样组织到一起都是无关紧要的。用专用名词来说，我们发现 E 的点形成了一个阿贝尔群，点 ∞ 是这个群的单位元素。

15.2 模 n 椭圆曲线

如果 n 是一个整数，我们可以用前面提到的方法对椭圆曲线进行模 n 计算。例如：考虑

$$E: y^2 \equiv x^3+2x+3 \pmod 5。$$

E 上的点都是满足该方程式的组合 $(x,y) \bmod 5$，包括无穷大的点。如下所示，这些可以被列出来，x 对 5 取模的可能结果为 0，1，2，3，4，把它们中的每一个代到方程式中，得到解方程式后 y 的值：

$$\begin{aligned}
&x \equiv 0 \Rightarrow y^2 \equiv 3 \pmod 5 \Rightarrow \text{无解}\\
&x \equiv 1 \Rightarrow y^2 \equiv 6 \equiv 1 \pmod 5 \Rightarrow y \equiv 1,4 \pmod 5\\
&x \equiv 2 \Rightarrow y^2 \equiv 15 \equiv 0 \pmod 5 \Rightarrow y \equiv 0 \pmod 5\\
&x \equiv 3 \Rightarrow y^2 \equiv 36 \equiv 1 \pmod 5 \Rightarrow y \equiv 1,4 \pmod 5\\
&x \equiv 4 \Rightarrow y^2 \equiv 75 \equiv 0 \pmod 5 \Rightarrow y \equiv 0 \pmod 5\\
&x = \infty \Rightarrow y = \infty。
\end{aligned}$$

因此，E 上的点为 $(1,1)$，$(1,4)$，$(2,0)$，$(3,1)$，$(3,4)$，$(4,0)$，(∞,∞)。

除了有理数 a/b 必须被作为 ab^{-1} 对待以外，在一个模 n 椭圆曲线上的点的加法可以通过前面给出的公式进行计算，其中 $b^{-1}b=1 \pmod n$，这要求 $\gcd(b,n)=1$ 且它是用椭圆曲线进行因数分解的密钥。

例：在刚才提到的曲线上计算 $(1,4)+(3,1)$。斜率为

$$m \equiv \frac{1-4}{3-1} \equiv 1 \pmod 5,$$

因此，

$$\begin{aligned}
x_3 &\equiv m^2-x_1-x_2 \equiv 1^2-1-3 \equiv 2 \pmod 5\\
y_3 &\equiv m(x_1-x_3)-y_1 \equiv 1(1-2)-4 \equiv 0 \pmod 5。
\end{aligned}$$

这意味着

$$(1,4)+(3,1)=(2,0)。$$

■

例：这里有一个稍微大一点的例子。令 $n=2773$，设

$$E: y^2 \equiv x^3+4x+4 \pmod{2773}, \text{且 } P=(1,3),$$

现在计算 $2P=P+P$。为了得到切线的斜率，我们在点 $(1,3)$ 处进行隐函数积分求值：

$$2y\,dy=(3x^2+4)\,dx \Rightarrow \frac{dy}{dx}=\frac{7}{6}。$$

但是我们正在模 2773 下计算，用扩展的欧几里得运算法则（参考 3.2 节）得到 $2311 \cdot 6 \equiv 1 \pmod{2773}$，所以能够用 2311 代替 1/6。因此，

$$m \equiv \frac{7}{6} \equiv 7 \times 2311 \equiv 2312 \pmod{2773}。$$

得到

$$x_3 \equiv 2312^2 - 1 - 1 \equiv 1771 \pmod{2773}$$
$$y_3 \equiv 2312(1 - 1771) - 3 \equiv 705 \pmod{2773}。$$

最终结果为

$$2P = P + P = (1771, 705)。$$

■

15.2.1　模 p 点的数目

令 $E: y^2 \equiv x^3 + ax + b \pmod{p}$ 表示一个椭圆曲线，其中 $p \geq 5$ 为素数，当 $x^3 + ax + b$ 为平方数模 p 的余数时，通过令 $x = 0, 1, \ldots, p-1$，可以列出 E 上的点。因为有一半的非零数是模 p 平方数，所以我们预测大约有一半的可能 $x^3 + ax + b$ 将是一个模 p 平方数的近似值。当它是一个非零平方数时，有两个平方根：y 和 $-y$。因此，大约有一半的可能得到两个 y 值，有一半的可能得不到 y 值。因此，我们预计大约有 p 个点，加上点 ∞，大约总计有 $p+1$ 个点。在 20 世纪 30 年代，H. Hasse 做出了这个更加精确的估计。

Hasse 定理。假设 $E \pmod{p}$ 有 N 个点，那么

$$|N - p - 1| < 2\sqrt{p}。$$

这个定理的证明远远超过了本书的范围。也可以证明，无论什么时候当 N 和 p 满足这个定理的不等式时，就存在一条确实有 N 个点的模 p 椭圆曲线 E 。

如果 p 比较大，约为 10^{20}，那么通过列举椭圆曲线上的点来计算它们是不现实的。Schoof，Atkin，Elkies 和其他人已经研究出了更精确的定理来解决这个问题。

15.2.2　基于椭圆曲线的离散对数

看看传统的离散对数问题：对于某个 k，已知 $x \equiv g^k \pmod{p}$，我们想找到 k。有一个椭圆曲线：假设在椭圆曲线 E 上有点 A，B，并且已知对于某个整数 k 有 $B = kA (= A + A + \cdots + A)$，我们想找到 k，这看起来不像一个对数问题，但很明显它是传统离散对数问题的相似问题，因此，它被称为椭圆曲线的**离散对数问题**。

关于椭圆曲线的离散对数问题一直没有一个好的解决方法。有一个 Pohlig-Hellman 方法的近似方法，在某些情况下有效。令 E 表示一个对素数 p 取模的椭圆曲线，n 表示满足 $nA = \infty$ 的最小整数，如果 n 仅仅有一个小的素数因子，那么计算离散对数 k 对素数的幂取模再除 n 的值，然后用中国剩余定理来得到 k（参考练习 12）是可能的。由于通过选择 E 和 A 不能实现 Pohlig-Hellman 方法，所以 n 有一个大的素数因子。

对 7.2 节描述的指数微积分方法没有什么可以替代的，这是因为没有一个好“小”近似体。你可能想设法用小坐标的点来替代小素数，但这是无用的。当你通过一个一个地分离素数因子来分解一个数字时，商变得越来越小，直到完成。在椭圆曲线中，可能有一个坐标相当小的点，删除一个小点，直到出现大坐标的点为止（参考上机题 5）。所以没有一个好的方法能够了解何时将在关于一些小点的因子基表示一个点上取得进步。

虽然在大多数情况下，关于离散对数的生日攻击在实际运用中要求太多的内存，但它对

椭圆曲线是有效的（练习 6）。想了解另外的方法，参考［Blake et al.］。

15.2.3 表示明文

在大多数密码体制中，必须有一种将原始信息映射为数字值的方法，依靠这种方法可以进行数学运算。为了运用椭圆曲线，我们需要一种把信息映射为椭圆曲线上某一点的方法。然后椭圆曲线密码体制对那一点采用椭圆曲线运算，生成一个表示密文的新的点。

把信息编码为椭圆曲线上的点的问题不像在传统情形下那样简单，特别地，不知道多项式的次数和用来记录任意椭圆曲线 $E(\bmod p)$ 上的点的确定性运算法则。虽然这样，通过快速概率法可以找到点，这些点可以用来对信息进行编码，这个方法在生成点时有小概率失败的特性，经过适当地选择参数，这个概率可以变得任意小，大约接近 $1/2^{30}$。

下面是 Koblitz 提出的一种方法，其思想如下所示，令 $E: y^2 \equiv x^3 + ax + b \pmod p$ 为椭圆曲线，消息 m（已经表示成数字）将被植入一个点的 x 坐标。然而，$m^3 + am + b$ 为模 p 平方数的概率仅仅为 1/2，因此，我们把 m 末尾的几位连接起来，并调节它们直到得到一个数字 x 满足 $x^3 + ax + b$ 为模 p 平方数。

更精确地，令 K 表示一个大整数，那么当设法把一条信息编码为点时，$1/2^k$ 的失败率是可以接受的。假设 m 满足 $(m+1)K < p$，消息 m 将用数字 $x = mK + j$ 表示，其中 $0 \leqslant j < K$。对 $j=0, 1, \ldots, K-1$，计算 $x^3 + ax + b$，并设法计算 $x^3 + ax + b \pmod p$ 的平方根。例如，如果 $p \equiv 3 \pmod 4$，那么就可以使用 3.9 节中的方法。如果有一个平方根 y，那么我们就认为 $P_m = (x, y)$；否则，把 j 加 1，用新的 x 值再试一次，重复上面的步骤直到找到一个平方根或 $j = K$。如果 j 始终等于 K，那么我们就不能把信息映射到一点，由于 $x^3 + ax + b$ 有大约一半的可能为一个模 p 平方数，所以大约有 $1/2^k$ 的可能会失败。

为了从点 $P_m = (x, y)$ 恢复信息，我们仅仅需要通过 $m = [x/K]$ 计算 m，其中 $[x/K]$ 表示小于等于 x/K 的最大整数。

例：令 $p = 179$，假设椭圆曲线为 $y^2 = x^3 + 2x + 7$，如果我们对 $1/2^{10}$ 的失败率感到满意，那么就可以接受 $K = 10$，因为需要 $mK + K < 179$，所以 $0 \leqslant m \leqslant 16$ 应该成立。假设消息 $m = 5$，我们考虑形式为 $mK + j = 50 + j$ 的 x，x 的可能选择为 50，51，...，59。对 $x = 51$，可以得到 $x^3 + 2x + 7 \equiv 121 \pmod{179}$ 和 $11^2 \equiv 121 \pmod{179}$，因此，用点 $P_m = (51, 11)$ 表示消息 $m = 5$，消息 m 能够通过 $m = [51/10] = 5$ 来恢复。 ■

15.3 用椭圆曲线因数分解

假设我们想对 $n = pq$ 进行因数分解，选择一个随机的模 n 椭圆曲线和曲线上的一点，实际上，我们一般选择几个含有点的（约 50 位的 14 个数，对大整数更多）曲线，并行地运行运算法则。

怎样来选择曲线呢？首先，选择一个点 P 和系数 a，然后选择 b，使 P 位于曲线 $y^2 = x^3 + ax + b$ 上，这比先选择 a 和 b 然后再找到一个点要有效的多。

例如：令 $n = 2773$，取 $P = (1, 3)$，$a = 4$，因为 $3^2 \equiv 1^3 + 4 \cdot 1 + b$，所以 $b = 4$。因此，

曲线为

$$E: y^2 \equiv x^3 + 4x + 4 \pmod{2773}。$$

在前面的例子中，我们计算过 $2P = (1771,705)$ 。注意，在计算过程中，需要找到 $6^{-1} \pmod{2773}$ 。这要求 $\gcd(6,2773) = 1$ ，并需要使用扩展的欧几里得运算法则，该法则本质上是一个最大公约数的计算。

现在让我们计算 $3P = 2P + P$ ，通过点 $2P = (1771,705)$ 和 $P = (1,3)$ 的直线有斜率 702/1770。当设法转换 1770 mod 2773 时，我们发现 $\gcd(1770,2773) = 59$，所以不能这样做，那么能做什么呢？我们的初始目标是要因数分解 2773，所以不必做任何其他的事情，我们已经找到了因子 59，该因子产生了因数分解 $2773 = 59 \cdot 47$。

下面就是所发生的事情。用中国剩余定理，我们能够把 E 当成一对椭圆曲线，一个对 59 取模，另一个对 47 取模，产生 $3P = \infty \pmod{59}$ ，而 $4P = \infty \pmod{47}$ 。因此，当我们设法计算 $3P$ 时，得到的斜率对 59 取模是无限的，对 47 取模是有限的，换句话说，有一个分母对 59 取模为 0，对 47 取模为非零数，计算 gcd 允许我们去分离因子 59。

许多因数分解的运算法则的基本原理都与此有相同的思想。如果 $n = pq$，那么只要它们行为表现相同，就不能将 p 和 q 分开，但是如果能够找到一些使它们行为表现稍微不同的东西，那么它们就能够被发现。在例子中，P 的倍数对 59 取模比对 47 取模更快地达到 ∞ ，因为通常素数 p 和 q 应该彼此相当独立地起作用，所以预计对于大多数曲线 $E \pmod{pq}$ 和点 P，P 的倍数对 p 取模和对 q 取模将在不同的时间达到 ∞ ，这将导致 gcd 要么是 p 要么是 q。

通常，比起 3 或 4 模 p 或模 q 达到 ∞ ，这里要花费更多的步骤。实际上，要用一个含有许多小素数因子的较大的数如 10000! 乘以 P，这可以通过连续的加倍来实现（连续自乘的加法替代；参考练习 10），希望 P 的倍数对 p 或对 q 取模为 ∞ ，这正是因数分解的 $p-1$ 法的替代法。虽然这样，由于当 $p-1$ 有一个大的素数因子时，$p-1$ 法（参考 6.4 节）经常无效。所以当满足 mP 等于 ∞ 的 m 有较大的素数因子时，同样的问题也会在刚刚描述的椭圆曲线方法中产生，如果这个问题发生了（于是这个方法不能马上产生一个因子），我们仅仅转到了一个新的曲线 E，这个曲线将与前面的曲线是相互独立的，满足 $mP = \infty$ 的值 m 在本质上应该与前面的 m 没有关系。尝试几次后（或者如果几个曲线被并行处理），常常会发现一条好的曲线，数字 $n = pq$ 也被因数分解，相反，如果 $p-1$ 法没有成功，除了使用一个不同的因数分解方法外，没有什么其他的办法。

例：我们想因数分解 $n = 455839$，选择

$$E: y^2 \equiv x^3 + 5x - 5 \text{ 和 } P = (1,1)。$$

假设我们设法去计算 $10!P$，有许多方法可以做，一种是计算 $2!P$，$3!P = 3\ (2!P)$，$4!P = 4\ (3!P)$，...，如果我们这样做，直到 $7!P$ 一切都很顺利，但是 $8!P$ 要求转化为 $599 \pmod{n}$，因为 $\gcd(599,n) = 599$ ，所以可以因数分解 n 为 599×761。

让我们更接近地看一下这个例子，计算显示 $E \pmod{599}$ 有 $640 = 2^7 \times 5$ 个点，$E \pmod{761}$ 有 $777 = 3 \times 7 \times 37$ 个点。此外，640 是满足在 $E \pmod{599}$ 上 $mP = \infty$ 的最小正数 m，777 是满足在 $E \pmod{761}$ 上 $mP = \infty$ 的最小正数 m。因为 8! 是 640 的倍数，很容易看到正如我们计算的那样在 $E \pmod{599}$ 上 $8!P = \infty$ ，因为 8! 不是 777 的倍数，所以满足在 $E \pmod{761}$ 上 $8!P \neq \infty$ 。前面提到过当除以 0 时得到 ∞ ，所以计算 $8!P$ 要求我们除以 $0 \pmod{599}$ ，这就是为什么能够找到因子 599。 ■

一般来说，考虑一个关于某个素数 p 的椭圆曲线 $E(\bmod p)$，在这个曲线上满足 $mP=\infty$ 的最小正数 m 能除尽 $E(\bmod p)$ 上点的数目 N（如果你知道群论，你可以把这看作是拉格朗日定理的一个推论），所以 $NP=\infty$。一般来说，m 将等于 N 或 N 的一个较大的约数，无论如何，如果 N 是一些小素数的乘积，那么 $B!$ 将是 N 的关于 B 的一个相当小的值的倍数。因此，$B!P=\infty$。

仅有小素数因子的数称为**平滑（smooth）**，更准确地说，如果一个整数的所有素数因子小于等于 B，那么它被称为 **B-平滑**。这个概念在二次方程式筛选法（6.4 节）、$p-1$ 因数分解法（6.4 节）和关于离散对数的指数微积分方法（7.2 节）中都占据着重要的地位。

回顾前面所讲的 Hasse 定理，如果 N 是一个接近于 p 的整数，很可能可以证明出如果平滑整数的密度足够大（这里我们将丢弃没有定义的“小”或“大”），如果我们选择一个随机的椭圆曲线 $E(\bmod p)$，那么数字 N 为平滑的可能性很大，这意味着椭圆曲线因数分解法应该通过曲线的选择而找到 p，如果我们尝试几条 $E(\bmod n)$ 曲线，其中 $n=pq$，那么很可能至少其中一条 $E(\bmod p)$ 或 $E(\bmod q)$ 上点的数目是平滑的。

简要地说，椭圆曲线因数分解法比 $p-1$ 法更优，因为 $p-1$ 法要求 $p-1$ 是平滑的，而椭圆曲线法仅仅要求有足够的平滑数接近于 p，所以随机选择的接近于 p 的整数中至少有一个是平滑的，这表明椭圆曲线因数分解法比 $p-1$ 法更容易成功。

椭圆曲线法似乎最适合分解中等大小的数字，阿拉伯数字 40 或 50 位左右，这些数字不再用于如 RSA 这样的基于因数分解的体制的安全中，但是有时在其他情况下有一个对于这些数字的快速的因数分解方法是非常有用的。对于更大的数字，二次方程式筛选法和数域筛选法更好一些（参考 6.4 节）。

退化曲线

实际上，三次多项式 x^3+ax+b 有重根的情形从来没有出现过。但是，如果它出现了将会发生什么呢？因数分解运算法则仍然有效吗？当且仅当有重根（这是当且仅当 $b^2-4ac=0$ 时 ax^2+bx+c 有重根的三次近似体）时，判别式 $4a^3+27b^2$ 为 0，因为我们正工作在模 $n=pq$ 下，所以结论为当且仅当判别式为 0 mod n 时有一个模 n 重根，由于 n 是合数，所以有 n 的最大公约数和判别式既不为 1 也不为 n 的中间情形，但这给出了一个 n 的非平凡因子，所以在本例中我们可以马上停下来。

例：让我们看一个示例：

$$y^2=x^3-3x+2=(x-1)^2(x+2)。$$

已知这个曲线上的一个点 $P=(x,y)$，结合数

$$(y+\sqrt{3}(x-1))/(y-\sqrt{3}(x-1))。$$

可以证明在曲线上增加点与乘上相应的数是对应的，只要我们不使用点（1，0），公式就有效，这是从哪里得到的呢？曲线在点（1，0）处的两条切线为 $y+\sqrt{3}(x-1)=0$ 和 $y-\sqrt{3}(x-1)=0$，这个数字仅仅是这两个表达式的比率。

因为我们需要工作在模 n 下，给出一个模 143 的例子。因为 3 为一个平方数对 143 取模的余数，所以选择 143；事实上，$82^2\equiv 3 \pmod{143}$。如果不是这种情况，这条曲线的情形将变得更加技术化，通过选择一条新的曲线可以很容易地调整这种情况。

考虑 $y^2=x^3-3x+2 \pmod{143}$ 上的点 $P=(-1,2)$，看看它的倍数：

$$P=(-1,2),\quad 2P=(2,141),\quad 3P=(112,101),\quad 4P=(10,20)。$$

当计算 $5P$ 时，我们发现了 143 的因子 11。

记得我们正在为曲线上的每一个点赋值，而不仅仅是点（1，1）。因为我们正工作在模 143 下，所以用 82 代替$\sqrt{3}$，与 $(-1,2)$ 相应的数是$(2+82(-1-1))/(2-82(-1-1)) \equiv 80 \pmod{143}$。对上面所有的点，我们可以计算这些数：

$$P \leftrightarrow 80, \quad 2P \leftrightarrow 108, \quad 3P \leftrightarrow 60, \quad 4P \leftrightarrow 81。$$

让我们与对 143 取模的 80 的幂运算进行比较：

$$80^1 \equiv 80, \quad 80^2 \equiv 108, \quad 80^3 \equiv 60, \quad 80^4 \equiv 81, \quad 80^5 \equiv 45。$$

我们得到了同样的数。这仅仅是前面提到的事实：曲线上增加点与乘以相应的数是对应的。此外，注意到 $45 \equiv 1 \pmod{11}$，但不是 mod 13，这与 5 乘以点（-1，2）为∞ mod 11 而不是 mod 13 的事实是相对应的。注意，1 为模 11 乘法的乘法单位，而∞为曲线上加法的加法单位。

从前面很容易看出，采用曲线 $y^2=x^3-3x+2$ 的因数分解法在本质上与采用传统的 $p-1$ 因数分解法是相同的（参考 6.4 节）。■

在前面的例子中，三次方程有一个二重根，更坏的可能性是三次方程有一个三重根。考虑曲线

$$y^2 = x^3,$$

对曲线上 $(x,y)\neq(0,0)$ 的点，结合数 x/y，从点 $P=(1,1)$ 开始，计算它的倍数：

$$P=(1,1), \quad 2P=\left(\frac{1}{4},\frac{1}{8}\right), \quad 3P=\left(\frac{1}{9},\frac{1}{27}\right),\cdots, \quad mP=\left(\frac{1}{m^2},\frac{1}{m^3}\right)。$$

注意到相应的数 x/y 为 1，2，3，...，m，在曲线上增加点与加上数 x/y 是相应的。

如果用曲线 $y^2=x^3$ 来因数分解 n，那么我们需要把点 mP 转变为模 n 整数，这要求找到 m^2 和 m^3 mod n 的逆元素，可以通过扩展的欧几里得定理来实现，该定理本质上是一个最大公约数的计算，当 $\gcd(m,n)\neq 1$ 时，可得到 n 的一个因子。因此，这种方法本质上与连续地计算 $\gcd(2,n)$，$\gcd(3,n)$，$\gcd(4,n)$，... 直到找到一个因子的方法是相同的，这是试探除法——众所周知的最老的因数分解法的一个慢速版本，当然，在椭圆曲线因数分解法中，P 的一个较大的倍数（$B!$）P 经常被计算，这与通过计算 $\gcd(B!,n)$ 来因数分解是等同的，它是一种经常用来对直到 B 的素数因子进行测试的方法。

简要地说，如果我们允许退化曲线，那么可知 $p-1$ 法和试探除法都包括在椭圆曲线的因数分解运算法则中。

15.4 特征为 2 的椭圆曲线

许多应用使用模 2 椭圆曲线或定义在有限域 $GF(2^n)$ 上的椭圆曲线（这些在 3.10 节中有描述），这是因为模 2 很适合计算机。

如果我们正工作在模 2 方式下，那么椭圆曲线的方程需要稍微地修改。原因有许多方面，例如，因为 2 与 0 是相同的，所以 y^2 的导数为 $2yy'=0$，这意味着所计算的切线是垂直的，所以对于所有的点 P，$2P=\infty$。一个更合理的解释是曲线 $y^2\equiv x^3+ax+b \pmod 2$ 有间断点（关于 x、y 的偏导数同时都不存在的点）。

我们需要

$$E: y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

这种形式的方程，其中 a_1，...，a_6是常量，加法定律稍微复杂一些，当且仅当三点位于同一直线时，才能使它们增加到无穷大，通过∞的直线也是垂直的。但是，正如在下面的例子中将要看到的那样，从 P 得到 $-P$ 与以前是不同的。

例：令 $E: y^2 + y \equiv x^3 + x \pmod 2$，同前面一样，我们可以列出 E 上的点：

$$(0,0),(0,1),(1,0),(1,1),\infty。$$

下面来计算 $(0,0)+(1,1)$。通过这两点的直线为 $y=x$，代入 E 的方程中得到 $x^2+x \equiv x^3+x$，这个等式可重写为 $x^2(x+1) \equiv 0$，根为 $x = 0,0,1 \pmod 2$。因此，相交的第三个点也是 $x=0$，因为它位于直线 $y=x$ 上，所以它必为 $(0,0)$（这可能令人迷惑。因为发生了这样的事情：直线在 $(0,0)$ 点与 E 相切，而在 $(1,1)$ 点与 E 相交)。和以前一样，现在得到

$$(0,0)+(0,0)+(1,1)=\infty。$$

为了得到 $(0,0)+(1,1)$，我们需要计算 $\infty-(0,0)$，这意味着我们需要找到满足 $P+(0,0)=\infty$ 的 P。通过∞的直线仍然是一条垂直线，在本例中，我们需要一条通过 $(0,0)$ 的直线，所以认为 $x=0$。它与 E 在点 $P=(0,1)$ 处相交，我们可推断出 $(0,0)+(0,1)=\infty$，把所有的加起来，得到

$$(0,0)+(1,1)=(0,1)。$$ ■

在大多数应用中，模 2 的椭圆曲线并不是足够大的，因此，定义在有限域的椭圆曲线经常被采用，对有限域的介绍请参考 3.10 节。虽然这样，在本节中，我们仅仅需要描述 $GF(4)$ 域。

令

$$GF(4)=\{0,1,\omega,\omega^2\},$$

遵守如下定律：

1. 对于所有的 $x, 0+x=x$。
2. 对于所有的 $x, x+x=0$。
3. 对于所有的 $x, 1\cdot x=x$。
4. $1+\omega=\omega^2$。
5. 加法和乘法满足交换律，结合律和分配律：对于所有的 x,y,z 有 $x(y+z)=xy+xz$。

因为

$$\omega^3=\omega\cdot\omega^2=\omega\cdot(1+\omega)=\omega+\omega^2=\omega+(1+\omega)=1,$$

可见 ω^2 是 ω 的乘法逆元素，因此，$GF(4)$ 的每一个非零元素都有一个乘法逆元素。

使用有限域系数的椭圆曲线与使用整数系数的椭圆曲线是同等对待的。

例：考虑

$$E: y^2+xy=x^3+\omega,$$

其中和以前一样 $\omega \in GF(4)$，用 $GF(4)$ 坐标列出 E 中的点：

$$x=0 \Rightarrow y^2=\omega \Rightarrow y=\omega^2$$

$$x=1 \Rightarrow y^2+y=1+\omega=\omega^2 \Rightarrow \text{无解}$$

$$x=\omega \Rightarrow y^2+\omega y=0 \Rightarrow y=1,\omega^2$$

$$x=\omega^2 \Rightarrow y^2+\omega^2 y=1+\omega=\omega^2 \Rightarrow \text{无解}$$

$$x = \infty \Rightarrow y = \infty 。$$

因此 E 上的点为

$$(0,\omega^2),(\omega,1),(\omega,\omega^2),\infty 。$$

我们来计算 $(0,\omega^2)+(\omega,\omega^2)$。通过这两点的直线为 $y=\omega^2$，把它代入 E 的方程：

$$\omega^4+\omega^2x = x^3+\omega,$$

得到 $x^3+\omega^2x=0$，有根 $x=0,\omega,\omega$。因此直线和 E 相交的第三点为 $(\omega,\ \omega^2)$，所以

$$(0,\omega^2)+(\omega,\omega^2)+(\omega,\omega^2)=\infty 。$$

我们需要 $-(\omega,\omega^2)$，也就是满足 $P+(\omega,\omega^2)=\infty$ 的点 P，垂直线 $x=\omega$ 与 E 相交于 $P=(\omega,1)$，所以

$$(0,\omega^2)+(\omega,\omega^2)=(\omega,1)。$$ ■

为了加密，椭圆曲线用在 $GF(2^n)$ 域，其中 n 很大，至少为 150。

15.5 椭圆曲线密码体制

椭圆曲线法存在于许多密码体制中，特别是那些包括离散对数的体制。工作在模 p 整数下的椭圆曲线有如下优点，在整数中，把因数分解运用在素数（特别是小素数）中来解决离散对数问题是可能的，正如指数微积分一样是众所周知的，在 7.2 节中已经描述过。对于椭圆曲线似乎没有好的近似体，因此，采用椭圆曲线使用较小的素数或较小的有限域而达到大的模 p 整数的安全级是有可能的，例如，这允许在硬件实现上有更大的存储。

下面，我们描述传统运算法则的三个椭圆曲线版本。正如我们将要看到的，把一个传统的基于离散对数的体制转变成用椭圆曲线的体制需要一个全面的过程：

1. 把模数乘运算转变为椭圆曲线上点的加法。
2. 把模数幂运算转换成用一个整数乘以曲线上的一个点。

当然，第二种情况实际上是第一种的特例，因为幂是由一个数字乘以它自身几次组成的，而用一个整数乘以一个点就是增加一个点到它自身几次。

15.5.1 椭圆曲线 ElGamal 密码体制

回忆一下前面的非椭圆曲线版本。艾丽斯想传送一个消息 x 给鲍勃，所以鲍勃选了一个大素数 p 和一个模 p 整数 α，他也选择了一个秘密的整数 a，并计算 $\beta\equiv\alpha^a \pmod p$。鲍勃公开 p，α，β，但把 a 保密，艾丽斯选择一个随机数 k，并计算 y_1 和 y_2，其中

$$y_1\equiv\alpha^k \text{ 和 } y_2\equiv x\beta^k \pmod p)。$$

她传送 (y_1,y_2) 给鲍勃，然后鲍勃通过计算

$$x\equiv y_2y_1^{-a} \pmod p$$

来解密。

现在描述椭圆曲线版本，鲍勃选择一个椭圆曲线 $E \pmod p$，其中 p 是一个大素数，他选择 E 上的一个点 α 和一个秘密的整数 a，他计算

$$\beta = a\alpha(=\alpha+\alpha+\cdots+\alpha),$$

这些点 α 和 β 是公开的，而 a 保密，艾丽斯用 E 上的一个点 x（参考 15.2 节）表示她的消息，她选择一个随机整数 k，计算

$$y_1 = k\alpha \text{ 和 } y_2 = x + k\beta,$$

并传送 y_1，y_2 给鲍勃。鲍勃通过计算

$$x = y_2 - ay_1$$

来解密。

这个体制的一个更有效的版本是由 Menezes 和 Vanstone 提出的，在 [Stinson, p. 189] 中有描述。

例：首先必须生成一个曲线，采用素数 $p = 8831$，点 $G = (x,y) = (4,11)$ 和 $a = 3$。为了使 G 位于曲线 $y^2 \equiv x^3 + ax + b \pmod p$ 上，我们接受 $b = 45$。艾丽斯有一条用点 $P_m = (5, 1743)$ 表示的消息，她希望传送给鲍勃，下面是她所做的。

鲍勃已经选择了一个随机数字 $a_B = 3$，并公开了点 $a_BG = (413,1808)$。

艾丽斯下载这个，并选择一个随机数 $k = 8$，她把 $kG = (5415,6321)$ 和 $P_m + k(a_BG) = (6626,3576)$ 传送给鲍勃，他首先计算 $a_B(kG) = 3(5415,6321) = (673,146)$，再从 (6626, 3576) 中减去这个：

$$(6626,3576) - (673,146) = (6626,3576) + (673, -146) = (5,1743)。$$

注意：点的相减采用了 15.1 节中的 $P - Q = P + (-Q)$ 规则。 ■

15.5.2 椭圆曲线 Diffie-Hellman 密钥交换

艾丽斯和鲍勃想交换一个密钥，为了完成这个功能，他们对一条椭圆曲线 $E: y^2 \equiv x^3 + ax + b \pmod p$ 上的公共基点 G 达成一致，我们选择 $p = 7211$，$a = 1$，$G = (3,5)$，得到 $b = 7206$。艾丽斯随机地选择 N_A，鲍勃随机地选择 N_B，假设 $N_A = 12$，$N_B = 23$，他们保持这些为自己私有，但公开 N_AG 和 N_BG。在本例中，有

$$N_AG = (1794,6375) \text{ 和 } N_BG = (3861,1242)。$$

艾丽斯现在接受 N_BG，并用 N_A 与之相乘得到密钥：

$$N_A(N_BG) = 12(3861,1242) = (1472,2098)。$$

同样地，鲍勃接受 N_AG，并用 N_B 与之相乘得到密钥：

$$N_B(N_AG) = 23(1794,6375) = (1472,2098)。$$

注意：他们有相同的密钥。

15.5.3 ElGamal 数字签名

用椭圆曲线来替代 8.2 节中描述的过程。为了说明我们正在对整数和椭圆曲线上的点进行操作的事实，有必要进行一些修改。

艾丽斯想对一条消息 m（这实际上可能是一个长信息的杂乱信号）进行签名，假设 m 是一个整数，她选定一条椭圆曲线 $E \pmod p$ 和 E 上的一个点 A，其中 p 为一个大素数，可以计算出来 E 上点的数目 n（参考练习 8）。假设 $0 \leqslant m < n$（如果不满足，选择一个大的 p），艾丽斯也选择一个私有整数 a，计算 $B = aA$。素数 p、曲线 E、整数 n、点 A 和 B 是公开的，

为了对这条信息签名，艾丽斯按如下步骤操作：

1. 选择一个满足 $1 \leqslant k < n$ 和 $\gcd(k,n) = 1$ 的随机整数 k，计算 $R = kA = (x,y)$。
2. 计算 $s \equiv k^{-1}(m - ax) \pmod n$。
3. 传送签名的消息（m，R，s）给艾丽斯。

注意：R 是 E 上的一个点，m 和 s 是整数。

鲍勃按如下的步骤验证签名：

1. 下载艾丽斯的公共信息 p，E，n，A，B。
2. 计算 $V_1 = xB + sR$ 和 $V_2 = mA$。
3. 如果 $V_1 = V_2$，证明签名是正确的。

因为

$$V_1 = xB + sR = xaA + k^{-1}(m - ax)(kA) = xaA + (m - ax)A = mA = V_2,$$

所以确认过程是有效的。

应该提到一个细节，在这个确认方程中，我们已经用满足 $k^{-1}k \equiv 1 \pmod n$ 的 k^{-1} 作为模 n 的整数，因此 $k^{-1}k$ 不为 1，而是一个与 1（mod n）同余的整数，所以对于某个整数 t，有 $k^{-1}k = 1 + tn$。可以证明 $nA = \infty$，因此，

$$k^{-1}kA = (1 + tn)A = A + t(nA) = A + t\infty = A。$$

当隐含了上面的假设后，上式证明了在确认方程式中 k^{-1} 和 k 互相抵消了。

传统的 ElGamal 方案和目前的椭圆曲线法都是彼此的近似体。模 p 整数被椭圆曲线 E 替代了，数 $p-1$ 变成了 n。注意，传统方法中的计算是对非零的模 p 整数进行操作，并且有 $p-1$ 个这样的同余类，椭圆曲线法对椭圆曲线上为 A 的倍数的点进行操作，这些点的数目是 n 的一个约数。

椭圆曲线法中 R 的 x 坐标的使用是任意的，给曲线上的点指定整数的任何方法都是有效的，用 x 坐标是一个容易的选择。同样，在传统方案中，在关于 s 的模 $p-1$ 的方程中整数 r 的使用似乎有点不自然，因为 r 最初是定义在模 p 方式下的，虽然这样，任何分配整数到模 p 整数的方法都是有效的，r 自身的使用是一个容易的选择（参考练习 8.9）。

存在一个与前述类似的数字签名运算法则的椭圆曲线版本（练习 11）。

15.6 习 题

1. 令 $x^3 + a_2x^2 + a_1x + a_0$ 表示一个根为 r_1，r_2，r_3 的三次多项式，证明 $r_1 + r_2 + r_3 = -a_2$。
2. (a) 列出椭圆曲线 E：$y^2 \equiv x^3 - 2 \pmod 7$ 上的点。
   (b) 在 E 上找到 $(3,2) + (6,5)$ 的和。
   (c) 在 E 上找到 $(3,2) + (3,2)$ 的和。
3. 通过选择一条随机的椭圆曲线和一个随机的点（如果无效，试另一个），用椭圆曲线法因数分解 $n = 11413$。
4. 假设想用椭圆曲线法来因数分解一个复合整数 n，从曲线 $y^2 = x^3 - 4x \pmod n$ 和点 $(2, 0)$ 开始，为什么不能生成 n 的因数分解？
5. 用椭圆曲线设计一个练习 7.4 (a) 中所用过程的近似体。

6. 证明怎样用生日攻击（参考 8.4 节）来解决椭圆曲线上的离散对数问题。

7. 证明：如果 P，Q，R 是椭圆曲线上的点，那么

$$P + Q + R = \infty \Leftrightarrow P,Q,R \text{ 在一条直线上。}$$

8. 令 P 为椭圆曲线 $E \bmod n$ 上的一个点。

(a) 证明由于 E 上有有限多个点，所以 P 有有限个不同的倍数。

(b) 证明存在满足 $i>j$ 且 $iP=jP$ 的整数 i，j，推断出 $(i-j)P=\infty$。

(c) 满足 $kP=\infty$ 的最小正整数 k 称为 P 的阶（**order**），令 m 表示满足 $mP=\infty$ 的整数，证明 k 整除 m。（提示：模仿练习 3.9（c，d）的证明。）

(d)（对了解一些群论的人）用拉格朗日关于群论的定理证明 E 上点的数目是 P 的阶的倍数。（结合 Hasse 的定理，这给出了一种找到 E 上点的数目的方法。参考上机题 1 和 4。）

9. 令 P 表示椭圆曲线 $E \bmod n$ 上的一个点，假设已知满足 $kP=\infty$ 的一个正整数 k，想证明（或驳斥）k 是 P 的阶。

(a) 证明对于 k 的某个素数因子 p，如果 $(k/p)P=\infty$，那么 k 不是 P 的阶。

(b) 假设 $m \mid k$ 且 $1 \leq m < k$，对于 k 的某个素数约数 p，证明 $m \mid (k/p)$。

(c) 假设对于 k 的每一个素数因子 $(k/p)P \neq \infty$，用练习 8（c）来证明 P 的阶为 k。（与练习 3.10 比较，具体例子参考上机题 4。）

10. (a) 令 $x=b_1b_2 \ldots b_w$ 代表二进制表示的整数，令 P 为椭圆曲线 E 上的一个点，执行下列过程：

(1) 首先令 $k=1, S_0=\infty$。

(2) 如果 $b_k=1$，令 $R_k=S_k+P$，如果 $b_k=0$，令 $R_k=S_k$。

(3) 令 $S_{k+1}=2R_k$。

(4) 如果 $k=w$，停止。如果 $k<w$，k 加 1，然后转到步骤（2）。

证明 $Rw=xP$。（与练习 3.12（a）比较。）

(b) 令 x 表示一个正整数，令 P 表示椭圆曲线上的一个点，证明下列过程可计算 xP。

(1) 开始 $a=x$，$B=\infty$，$C=P$。

(2) 如果 a 是偶数，令 $a=a/2$，$B=B$，$C=2C$。

(3) 如果 a 是奇数，令 $a=a-1$，$B=B+C$，$C=C$。

(4) 如果 $a \neq 0$，转到步骤 2。

(5) 输出 B。

（与练习 3.12（b）比较。）

11. 这里是一个数字签名算法的椭圆曲线版本，艾丽斯想对消息 m 签名，m 是一个整数，她选择一个素数 p，一条椭圆曲线 $E\ (\bmod p)$，计算出了 E 上点的数目 n，发现了 n 的一个大素数因子 q，选择了满足 $qA=\infty$ 的一个点 $A\ (\neq\infty)$。（事实上，n 是不必要的，选择 E 上的一个点 A'，找出满足 $mA'=\infty$ 的整数 m，虽然不容易，但有些方法可以实现，令 q 表示 m 的一个大素数因子，如果存在，令 $A=(m/q)A'$，那么 $qA=\infty$。）假设 $0 \leq m < q$，艾丽斯选择她的秘密整数 a 并计算 $B=aA$，公共信息为 p，E，q，A，B，艾丽斯进行如下操作：

(1) 选择满足 $1 \leq k < q$ 的随机整数 k，计算 $R=kA=(x, y)$；

(2) 计算 $s \equiv k^{-1}(m+ax) \pmod q$；

(3) 传送签名的信息 (m, R, s) 给鲍勃。

鲍勃如下所示验证签名：

(1) 计算 $u_1 \equiv s^{-1}m \pmod q$ 和 $u_2 \equiv s^{-1}x \pmod q$；

(2) 计算 $V = u_1A + u_2B$；

(3) 如果 $V=R$，那么宣布签名正确。

(a) 证明验证方程拥有一个正确的签名信息，$qA=\infty$ 在什么地方使用（参考 15.5 节在 ElGamal 方案中提到的“细小的点”）？

(b) 为什么 $k^{-1} \pmod q$ 存在？

(c) 如果 q 比较大，那么为什么 $s^{-1} \pmod q$ 不存在的概率很小？当它不存在时我们怎样验证这个例子？（当然，在本例中，艾丽斯应该选择一个新的 k 重新开始。）

(d) 这里在验证过程中进行了多少次“（大整数）×（E 上的点）”的计算？本文描述的椭圆 ElGamal 方案的验证过程进行了多少次这样的计算（参考 8.5 节的末尾）？

12. 令 A 和 B 代表椭圆曲线上的点，假设对于任意的整数 k，$B=kA$，同样假设对于某个整数 n，$2^nA=\infty$，但是 $T=2^{n-1}A\neq\infty$。

(a) 证明：如果 $k\equiv k' \pmod{2^n}$，那么 $B=k'A$，因此我们可以假设 $0\leqslant k<2^n$。

(b) 令 j 表示一个整数，证明当 j 为偶数时 $jT=\infty$，当 j 为奇数时 $jT\neq\infty$。

(c) 令 $k=x_0+2x_1+4x_2+\cdots+2^{m-1}x_{m-1}$，其中每个 x_i 为 0 或 1（k 的二进制扩展），证明当且仅当 $2^{n-1}B=\infty$ 时 $x_0=0$。

(d) 假设对于某个 $m<n$，已知 $x_0,\ldots,x_{m-1}$，令 $Qm=B-(x_0+\cdots+2^{m-1}x_{m-1})A$，证明当且仅当 $x_m=0$ 时，$2^{n-m-1}Q_m=\infty$。这允许我们找到 x_m，继续用这种方法，可以得到 $x_0,\ldots,x_{n-1}$，因此可以计算 k。这个技巧可以被扩展到 $sA=\infty$ 的例子中，其中 s 为仅有小素数因子的整数，这是 Pohlig – Hellman 运算法则的近似体（参考 7.2 节）。

15.7 上 机 题

1. 令 E 表示椭圆曲线 $y^2\equiv x^3+2x+3 \pmod{19}$。

(a) 找出 $(1,5)+(9,3)$ 的和。

(b) 找出 $(9,3)+(9,-3)$ 的和。

(c) 用 (b) 中的结果找出 $(1,5)-(9,3)$ 的差值。

(d) 找到满足 $k(1,5)=(9,3)$ 的整数 k。

(e) 证明 $(1,5)$ 确实有 20 个不同的倍数，包括 ∞。

(f) 用 (e) 和练习 8 (d)，证明 E 上点的数目为 20 的倍数，用 Hasse 定理证明 E 确实有 20 个点。

2. 想用曲线 $y^2\equiv x^3+7x+11 \pmod{593899}$ 上的点 (x,y) 表示信息 12345，令 $x=$ 12345 _，找到 x 丢失的最后一个数字的值，其中 x 为椭圆曲线上某点的 x 坐标。

3. (a) 用椭圆曲线因数分解 3900353。

(b) 用 6.4 节的 $p-1$ 法因数分解 3900353。用从(a) 中得到的关于素数因子的信息，解释为什么 $p-1$ 法对解决这个问题效果不是很好？

4. 令 $p = (2,3)$ 表示椭圆曲线 $y^2 \equiv x^3 - 10x + 21 \pmod{557}$ 上的一个点。

(a) 证明 $189P = \infty$，而 $63P \neq \infty$，$27P \neq \infty$。

(b) 用练习 9 证明 P 有阶 189。

(c) 用练习 8 (d) 和 Hasse 的定理证明该椭圆曲线有 567 个点。

5. 计算关于椭圆曲线 $y^2 \equiv x^3 - 11x + 11 \pmod{593899}$ 的差值$(5,9) - (1,1)$。注意，即使初始点的坐标很小，答案中也包括大的整数。

第 16 章 纠错码

在一个好的密码体制中，密文中一个比特的修改将改变明文中很多比特，以致于明文难以理解，因此我们需要一种方法来检测并纠正密文在传输中可能发生的错误。

在许多非密码体制的环境中也需要用到纠错，比如传真机、计算机硬件设备、CD 播放机以及其他任何用数字描述数据的设备，纠错码可以解决这个问题。

尽管编码理论（噪声信道中的通信）在学术上不是密码学（非保密信道中的通信）的一部分，在本章的 16.1 节我们将描述怎样用纠错码建立公钥密码体制。

16.1 绪　　论

所有的信道都会有某种程度的噪音，也就是由各种干扰源产生的干扰，比如临近的信道、电子脉冲、设备引起的衰减等。这些噪音能够干扰数据的传输，就像在一个吵闹的房间里交谈，噪音越大交谈越困难，同样随着信道噪音的变大在信道中传输数据将变得更加困难。为了能够在一个嘈杂的房间里正常交谈，要么提高你的音量，要么不断地被迫重复。第二种方法是我们所关心的，也就是说我们必须在传输中增加一些冗余，以保证接受者能够重建收到的信息。下面给出我们将会用到的一些技术实例，在每一个实例中原始信息中的记号将被包含冗余的码字(*codewords*) 代替。

例 1：循环码

考虑一个字母表 $\{A,B,C,D\}$ ，我们要在误码率为 $p = 0.1$ 的噪声信道中传输一个字母，如果想要传输字母 C，那么接收到的符号为 C 的概率是 90%，这势必带来太大的错误概率。或者我们将字母 C 重复三次，即 CCC，假设有一个错误发生并且接收到的消息是 CBC，我们选择发生频率最大的符号作为消息，即字母 C，那么正确接收的概率是三个字母都正确的概率加上只有一个字母错误的概率：

$$(0.9)^3 + 3(0.9)^2(0.1) = 0.972,$$

这样就大大减小了错误的概率。

编码的两个重要的概念是检错和纠错，如果最多有两个错误发生，这种纠错码就能够检测到错误的发生，如果收到的消息是 CBC，那么可能是 CCC 发生了一个错误或者是 BBB 发生了两个错误，但不能确定是哪一种情况；如果最多有一个错误发生，那么我们能够纠正错误并推断发送的消息是 CCC。如果将字母重复两次，那么能够检测到错误但不能纠正它(收到 CB 能判断发送的是 CC 还是 BB 吗?)。

这个例子指出纠错码能够使用任何符号集，一般而言，使用的符号是诸如整数、二进制串之类的数字。比如，我们能够用两比特的二进制串 00，01，10，11 代替字母 A，B，C，D。那么前面所述的过程（重复三次）将得到以下码字：

$$000000,010101,101010,111111。$$ ■

例 2：奇偶校验

假设我们要传送 7 比特的消息，增加第 8 个比特使非零比特位数为偶数。例如，发送的消息 0110010 变成 01100101，消息 1100110 变成 11001100。如果收到的信息含有奇数个非零位，那么传输中一个比特的错误立即可以被发现，但由于任何一位发生错误都能产生奇数个非零位，因此不能判断是哪一个比特错误。当检测到一个错误时最好的方法就是重传信息。 ■

例 3：二维奇偶码

上面例子中的奇偶校验码能够用来设计一种可以纠正一比特错误的代码，而二维奇偶校验码可以将数据排列到二维数组中，然后对每一行、每一列计算奇偶比特数。

为了演示这种编码方法，假设我们要对 20 比特的数据 10011011001100101011 进行编码，将 20 比特排列到如下所示的一个 4×5 矩阵中，并对行和列计算奇偶比特数。

$$\begin{matrix}1&0&0&1&1\\0&1&1&0&0\\1&1&0&0&1\\0&1&0&1&1\end{matrix}$$

通过计算所有列的奇偶数之和定义扩展矩阵右下角的比特位，这样就得到了下面的 5×6 矩阵。

$$\begin{matrix}1&0&0&1&1&1\\0&1&1&0&0&0\\1&1&0&0&1&1\\0&1&0&1&1&1\\0&1&1&0&1&1\end{matrix}$$

假设扩展矩阵被传输并且在第三行第四列有一个比特的错误发生，接收者将接收到的比特排列成下面的 5×6 矩阵。

$$\begin{matrix}1&0&0&1&1&1\\0&1&1&0&0&0\\1&1&0&1&1&1\\0&1&0&1&1&1\\0&1&1&0&1&1\end{matrix}$$

第三行、第四列的奇偶比特数为奇数，这样就可以判断错误发生在第三行第四列。

如果有两个错误发生，这种编码方法能够检测到错误，比如错误发生在第二行的第二、第三列上，那么对第二、第三列的奇偶校验将指示两个比特错误的存在。然而在这个实例中有很多种错误的可能，因而不能纠正错误。比如，如果第五行的第二、第三比特发生错误，那么奇偶校验将得到和第二行发生错误时相同的结果。 ■

例 4：汉明码（Hamming [7，4] code）

原始信息由包含 4 个比特位的二进制分组组成，用原始信息与包含 7 个比特位的二进制块（如下）相乘得到的编码代替原始信息。

$$G=\begin{pmatrix}1&0&0&0&1&1&0\\0&1&0&0&1&0&1\\0&0&1&0&0&1&1\\0&0&0&1&1&1&1\end{pmatrix}。$$

例如，消息 1100 变成：

$$(1,1,0,0)\begin{pmatrix}1&0&0&0&1&1&0\\0&1&0&0&1&0&1\\0&0&1&0&0&1&1\\0&0&0&1&1&1&1\end{pmatrix}\equiv(1,1,0,0,0,1,1)\pmod 2。$$

由于矩阵 G 的前 4 列为单位矩阵，因而输出的前 4 个比特就是原始信息，剩余的 3 个冗余比特提供了检错和纠错的能力。事实上，在 7 位的编码中如果只有一个错误，我们能够很容易地纠正这个错误。

假设传送的码字为 1100011，而收到的是 1100001，我们怎样才能检测并纠正错误呢？把矩阵 G 写成 $[I_4,P]$ 的形式，P 是一个 4×3 的矩阵，构造矩阵 $H=[P^T,I_3]$，其中 P^T 是 P 的转置矩阵，用收到的消息乘以 H 的转置矩阵：

$$(1,1,0,0,0,0,1)\begin{pmatrix}1&1&0&1&1&0&0\\1&0&1&1&0&1&0\\0&1&1&1&0&0&1\end{pmatrix}^T$$

$$\equiv(1,1,0,0,0,0,1)\begin{pmatrix}1&1&0\\1&0&1\\0&1&1\\1&1&1\\1&0&0\\0&1&0\\0&0&1\end{pmatrix}\equiv(0,1,0)\pmod 2。$$

结果是 H^T 的第六行，即在收到的消息中第六个比特位有一个错误，因此正确的码字应该是 1100011，前 4 个比特给出了原始信息 1100。如果没有错误，那么收到的消息与 H^T 相乘将得到（0，0，0），因此我们知道不需要纠错，这个过程我们还将在 16.5 节汉明码的内容中继续阐述。在这里我们只需要了解汉明码能够有效地纠正一个比特的误码。

汉明码［7，4］是循环码的重大改进，在汉明码中，如果要传递 4 个比特的信息，需要传送 7 个比特，这样我们可以检测到最多两个错误，纠正最多一个错误。而对于循环码，为了获得同样的检错和纠错能力，我们需要传送 12 个比特以达到传递 4 个比特信息的目的。在后续章节中我们将用编码效率来阐述这种机制，汉明码的编码效率是 4/7，而循环码的编码效率是 $4/12\approx1/3$。一般而言，在保证纠错能力的情况下，编码效率越高越好，例如直接发送 4 个比特的消息，其编码效率为 1，但是由于不具备纠错能力，这种方式在任何条件下都不适用。 ■

例 5：ISBN 码

国际标准图书编码（ISBN）提供了另外一种纠错码的例子，国际标准图书编码是对出版的图书进行十进制的编码。例如一本书的 ISBN 数字是 0—13—061814—4，第一个数字代表这本书所用的语言；0 指英语。接下来的两位数字代表出版商；比如 13 代表你所读的这本书是 Prentice Hall 出版公司出版的。后面 6 位数字代表了出版商对这本书的编号，由于 ISBN 码采用的是十进制数字，因此 ISBN 数字 $a_1a_2\cdots a_{10}$满足：

$$\sum_{j=1}^{10} ja_j \equiv 0 \pmod{11},$$

这个等式是模 11 得到的，前 9 个数字 $a_1a_2\cdots a_9$取自集合 $\{0, 1, \cdots, 9\}$，而 a_{10}可能是 10，在这种情况下用符号 X 表示。

假设 ISBN 的数字 $a_1a_2\cdots a_{10}$通过一个噪声信道传播或者写在图书定单上，而收到的 ISBN 数字是 $x_1x_2\cdots x_{10}$。ISBN 码能够检测到一个错误或者由于数字置换引起的两位错误。为了实现检错，需要计算加权校验和：

$$S = \sum_{j=1}^{10} jx_j \equiv 0 \pmod{11}。$$

如果 $S \equiv 0 \pmod{11}$，我们不能检测到任何错误，尽管也存在发生一个错误没有检测到的可能。否则我们能检测到一个错误，但是不能纠错（参见练习 2）。

如果 $x_1x_2\cdots x_{10}$ 与 $a_1a_2\cdots a_{10}$ 只有一位 x_k 不同，假设 $x_k = a_k + e$ 而 $e \neq 0$，用下面的公式计算 S：

$$S = \sum_{j=1}^{10} ja_j + ke \equiv ke \pmod{11},$$

这样，如果有一个错误发生，我们就能够检测到。另外一种可以可靠地检测到错误的情况是 a_k和 a_l掉换了位置，这是人们抄写数字时最容易犯的错误。在这种情况下，$x_l = a_k$ 和 $x_k = a_l$，用下面的公式计算 S：

$$S = \sum_{j=1}^{10} jx_j = \sum_{j=1}^{10} ja_j + (k-l)a_l + (l-k)a_k \pmod{11}$$

$$\equiv (k-l)(a_l - a_k) \pmod{11}$$

如果 $a_l \neq a_k$，那么校验和就不为零，从而可以判断有一个错误。 ■

例 6：阿达玛编码（Hadamard code）

这种码第一次被水手号太空船用来向地球传送照片，这种码共有 64 个码字；32 个码字用 32 × 32 矩阵中的行表示

$$H = \begin{pmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & -1 & 1 & -1 & \cdots & -1 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & -1 & -1 & 1 & \cdots & -1 \end{pmatrix}。$$

下面是这个矩阵的构造方法，将矩阵的行和列从 0 到 31 进行编号，第 i 行和第 j 列的交点处为元素 $h_{i,j}$，将 i 和 j 写成 $i = a_4a_3a_2a_1a_0$ 和 $j = b_4b_3b_2b_1b_0$的二进制形式，那么

$$h_{ij} = (-1)^{a_0b_0+a_1b_1+\cdots+a_4b_4}。$$

例如，$i = 31$，$j = 3$，二进制表示为 $i = 11111$，$j = 00011$，因此 $h_{31,3} = (-1)^2 = 1$。

另外 32 个码字由 $-H$ 的行得到，在矩阵 H 中任意两行若相等，则这两行的点积为 32，

否则点积为 0。当水手号传送照片时每一个像素由 6 比特的数字表示，然后编码成 64 个码字之一进行传送，每个收到的信息（由 1 和 -1 组成的 32 位二进制数字串）能够按如下方式解码（即纠正为正确的码字），收到的信息与矩阵 H 的每一行进行点积，如果其中只有一行的点积值为 ±32，其他所有行的点积值都是 0，那么收到的消息是正确的。如果点积值是 32，那么码字就是矩阵 H 对应的行；如果点积值是 -32，那么码字就是矩阵 $-H$ 对应的行。如果信息有一个错误，那么计算收到的消息与矩阵 H 每一行的点积，其中只有一行的点积值为 ±30，其他所有行的点积值都是 ±2，同样我们可以得到发送的码字即为 ±30 对应的行。如果消息有两个错误，那么计算收到的信息与矩阵 H 每一行的点积，其中只有一行的点积值为 ±32，±30 或 ±28，其他所有行的点积值都是 0，±2，±4。以此类推，我们可以知道当发生 7 个错误时，除了有一个点积值在 -30 到 -16 之间或 16 到 30 之间以外，其他所有的点积值都在 -14 到 +14 之间，这样就可以得到正确的码字。

当发生 8 个或更多错误时，由于点积值发生了重叠，因而纠错是不可能的。但是由于只有当发生 16 个错误时才会由一个码字变为另一个码字，所以这种编码方法最多能够检测 15 个错误。

由于这种编码方法采用 32 个比特传送 6 比特的信息，因此其编码效率相对较低，只有 6/32。然而考虑到权衡高纠错率，由于水手号太空船传送的信号相当微弱，潜在的错误非常高，因此高的纠错率是必需的。另一种可能的选择是增加信号能量，采用编码效率较高、纠错能力较低的编码方法。然而，在这个实例中，事实证明能量的节省完全弥补了速率的损失，这个问题在特定的应用中考虑采用哪种编码方式时是一个重要的工程因素。

16.2 纠 错 码

一个信息发送器将信息**编码**（**encode**）成包含连续符号的码字，这些码字通过噪声信道传输给接收器，如图 16.1 所示。收到的符号序列经常会有错误，因此可能并不是最初发送的码字，接收器必须**解码**（**decode**），即通过纠错将收到的信息转换为码字，然后恢复成原始信息。

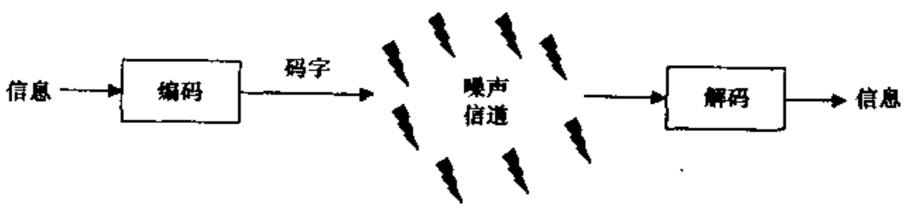


图 16.1 编码和解码

构成码字的符号属于一个字母表，当这个字母表由二进制位 0 和 1 组成时，这种码称为**二进制码**（**binary code**）。由包含 3 个符号的序列组成的码通常用模 3 整数表示，这种编码称为**三进制编码**（**ternary code**）。一般而言，一种由包含 q 个符号的序列组成的码被称为 **q 进制编码**（**q-ary code**）。

定义：假设 $\mathcal{A}$ 是一个字母表，$\mathcal{A}^n$ 描述了由 $\mathcal{A}$ 中元素组成的 n 维数组的集合，一个长度

为 n 的码是集合 $\mathcal{A}^n$ 的非空子集。

构成一个码的 n 维数组称为**码字（codewords）**或码矢。例如在一个二进制的循环码中，每个符号重复三次，字母表是集合 $\mathcal{A} = \{0,1\}$，且编码的集合 $\{(0,0,0),(1,1,1)\} \subset \mathcal{A}^3$。

严格来讲，这里定义的编码称为**分组码（block codes）**，其他的编码可能存在着变长的码字，这些编码方法将在本章的最后简要描述，这里我们只讨论分组码。

对于由 $\mathcal{A}^n$ 的随机子集构成的码，解码将是一个非常费时的过程，因此所有有用的码都是满足一些附加条件的 $\mathcal{A}^n$ 的子集。最常见的条件是要求 $\mathcal{A}$ 是一个有限域，因而 $\mathcal{A}^n$ 是一个向量空间，并且要求码是该向量空间的子空间，这种码被称为线性的（*linear*），我们将在 16.4 节中讨论。

在本章其他部分，我们将讨论任意的、可能是非线性的编码，通常假设这样的码字是 n 维的向量。

为了解码，找到一个度量两个向量之间距离的方法非常必要，汉明距离提供这样的方法。假设 u，v 是 $\mathcal{A}^n$ 中的两个向量，**汉明距离（Hamming distance）** $d(u, v)$ 是这两个向量之间不同元素的个数。例如，如果使用二进制向量，并且向量 $u = (1,0,1,0,1,0,1,0)$，向量 $v = (1,0,1,1,1,0,0,0)$，那么 u 和 v 有两位不同（第四位和第七位不同），因此最小汉明距离 $d(u,v) = 2$。另外一个例子，假设我们使用的是英语字母表，由于两个字符串 *fourth*，*eighth* 有 4 个位置不同，因而 $d(fourth, eighth) = 4$。

汉明距离 $d(u,v)$ 的重要性在于它提供了计算从 u 变为 v 发生的最小“错误”个数的方法。下面给出汉明距离的一些基本属性。

命题：属性 $d(u, v)$ 是集合 $\mathcal{A}^n$ 上的一个度量，它满足：

1. $d(u,v) \geqslant 0$ 并且当且仅当 $u = v$ 时 $d(u,v) = 0$。
2. 对于任意的 u,v，有 $d(u,v) = d(v,u)$。
3. 对于任意的 u,v,w，有 $d(u,v) \leqslant d(u,w) + d(w,v)$

第三个属性通常被称为三角不等式。

证明：(1) $d(u,v) = 0$ 等价于向量 u 和 v 没有差别，即 $u = v$。属性（2）是显而易见的。对于属性（3），如果 u 和 v 有一个位置不同，那么 u 和 w 的这个位置不同，或者 w 和 v 的这个位置不同，或者两者均成立，因此 u 和 v 不同位置的个数小于 u 和 w 不同位置的个数与 w 和 v 不同位置的个数之和。 □

对于编码 C，我们能够计算任意两个不同码字之间的汉明距离。除了这个距离表格之外，还有一个最小值 $d(C)$，称之为码 C 的**最小距离（minimum distance）**，换句话说，即：

$$d(C) = \min\{d(u,v) \mid u,v \in C, u \neq v\}$$

由于最小距离给出了将一个码字变成另一个码字需要的最小错误数，因而码 C 的最小距离是很重要的一个数字。

当一个码字通过一个噪声信道传输时，错误被引入到某些向量元素中，我们通过寻找与收到的向量的汉明距离最小的码字来纠错，换言之，我们通过尽量少的修改将收到的向量变为码字，这种方法称为**最小邻域解码（nearest neighbor decoding）**。

如果改变一个码字中 s 个位置而不能将其变为另一个码字，那么这种编码最多能检测到 s 个错误，如果改变一个码字 c 少于 t 个位置，其最近的码字仍然是 c，那么这种编码最多能

纠正 t 个错误。对于纠错，上面的定义没有提供有效的算法，它仅仅要求当误码小于等于 t 时，通过最小邻域解码得到正确的答案。下面是纠错码理论的一个重要结论。

定理：

1. 如果 $d(C) \geq s+1$，那么编码 C 最多能够检测 s 个错误。
2. 如果 $d(C) \geq 2t+1$，那么编码 C 最多能够纠正 t 个错误。

证明：(1) 假设 $d(C) \geq s+1$，如果码字 c 被发送，并且发生了不多于 s 个错误，那么收到的消息 r 不可能是另一个码字，因此可以检测到错误。

(2) 假设 $d(C) \geq 2t+1$，如果码字 c 被发送，而且收到的消息 r 发生了不多于 t 个错误，即 $d(c,r) \leq t$，如果 c_1 是 c 之外的任意一个码字，我们断言 $d(c_1,r) \geq t+1$。为了证明该断言，我们假设 $d(c_1,r) \leq t+1$，然后将之代入三角不等式中，得到

$$2t+1 \leq d(C) \leq d(c,c_1) \leq d(c,r) + d(c_1,r) \leq t+t = 2t。$$

这是一个永假式，因此 $d(c_1,r) \geq t+1$。由于 r 有不多于 t 个错误，所以最小邻域解码法能将 r 解码为 c。 □

怎样才能找到最近的邻域呢？一种方法是计算接收的消息 r 和每一个码字的距离，然后选择汉明距离最小的码字。在实践中，对于大量的编码这种方法是不切实际的，一般而言解码是复杂的，而且有相当多的探索工作都在致力于寻找一个快速的解码运算法则。在下面的章节里，我们将讨论一些特殊类型编码对应的解码技术。

在继续讨论之前，我们先介绍一些必要的记号。

记号： 一种长度为 n、有 M 个码字、最小距离 $d=d(C)$ 的编码记作 **(n, M, d) 码**。

当我们讨论线性编码时，有一个类似的记号 $[n,k,d]$ 码，请注意后一个记号使用的是方括号而前一个使用的是圆括号（这两个记号并不像人们以为的那样容易混淆）。

二进制循环码 $\{(0,0,0),(1,1,1)\}$ 是一种 (3, 2, 3) 码，16.1 节例 6 中的阿达玛 (Hadamard) 码是 (32, 64, 16) 码（因为 $16 \geq 2 \cdot 7+1$，它最多能够纠正 7 个错误）。

如果我们有一个 q 进制 (n, M, d) 码，那么用下面的公式定义**编码率 (code rate)** 或**信息率 (information rate)** R：

$$R = \frac{\log_q M}{n}$$

例如，对于阿达玛码，$R = \log_2(64)/32 = 6/32$。编码率代表了输入的数据符号数与传输的编码符号数的比值，由于编码率代表了在传输真正数据时所使用的带宽，因此编码率在实际的系统中是一个重要的参数。在 16.1 节的例 4 和例 6 中提到了编码率，在 16.3 节中我们将讨论编码率的一些限制。

给定一个编码，我们能够构造本质上相同的其他编码。假设有一个码字 c 表示为 $c = (c_1,c_2,\cdots,c_n)$，那么可以通过置换 c 中元素的次序来定义 c 的一个变换，例如新的向量 $c' = (c_2,c_3,c_1)$ 是码字 $c = (c_1,c_2,c_3)$ 的一个可能的置换。另外一种操作是符号变换，假设对 q 进制的符号采取一个符号变换，我们将对每一个码字的对应位置实行这个置换，如果对三进制采取下面的置换 $\{0\to2, 1\to0, 2\to1\}$，有如下 3 个码字：(0, 1, 2) (0, 2, 1) 和 (2, 0, 1)，然后对所有码字的第二个元素执行上述变换得到如下的向量：(0, 0, 2)，(0, 1, 1) 和 (2, 2, 1)。

如果一种编码能够通过下面的一系列操作得到另一种编码，则我们正式地称这两种编码

是等价（equivalent）的。

1. 交换编码的位置；

2. 置换所有码字中固定位置的符号。

很明显所有与（n，M，d）码等价的编码也是（n，M，d）码。然而，对于同样的 n，M，d 可能有很多不等价的（n，M，d）码。

16.3 一般编码的边界条件

我们已经知道如果 $d \geq 2t+1$，则一个（n，M，d）码能够纠正 t 个错误，因此希望最小距离 d 取值尽可能大，这样可以纠正更多的错误。但是我们也希望 M 取值更大，这样可使编码效率尽量接近于 1。编码率越接近于 1，在噪声信道中传输信号时能够更有效地利用带宽，然而增加 d 将会增加 n 或减少 M。

在本节中我们将研究 n，M 和 d 之间的关系，而不关心实践方面的因素，比如不考虑是否用好参数的编码能够得到高效的解码算法之类的问题。尽管这种研究是有益的，它能够告诉我们，由于理论的限制实际的编码能够达到什么效果。

首先我们根据 n 和 d 考虑 M 的上边界条件，然后证明存在 M 比下边界大的编码，最后讨论一些例子和这些边界条件的关系。

16.3.1 上边界条件

我们的第一个结论是由 R. Singleton 在 1964 年提出的，通常称为 **Singleton 边界**。

定理：假设 C 是一个 q 进制（n，M，d）码，那么

$$M \leq q^{n-d+1}。$$

证明：对于码字 $c=(a_1,\ldots,a_n)$，设 $c'=(a_d,\ldots,a_n)$，如果 $c_1 \neq c_2$ 是两个不同的码字，则它们至少有 d 个位置不同。由于 c_1' 和 c_2' 是由 c_1，c_2 去掉 $d-1$ 个元素得到的，因此它们至少有一个位置不同，即 $c_1' \neq c_2'$。因此码字 c 中 M 的值等于通过该方法获得的向量 c' 的个数，由于在向量 c' 中有 $n-d+1$ 个位置，因此最多有 q^{n-d+1} 个向量 c'，这意味着 $M \leq q^{n-d+1}$。得证。 □

推论：q 进制（n，M，d）码的编码率最大为 $1-\frac{d-1}{n}$。

证明：由编码率的定义直接可以得到推论。 □

推论指出，**相对最小距离（relative minimum distance）**d/n 越大，编码率越小。

一个编码以等号满足 Singleton 边界条件时称为 **MDS 码（maximum distance separable）**。这个 Singleton 边界能够写成 $q^d \leq q^{n+1}/M$，因此对给定的 n 和 M，一个 MDS 码有可能的最大 d 值。Reed-Solomon 码（见 16.9 节）是 MDS 码中重要的一类编码。

在推导另一个边界条件之前，先介绍一些在纠错码中要用到的几何表示。以码字 c 为中心、半径为 t 的汉明球（**Hamming sphere**）表示为 $B(c,t)$，它被定义为所有到码字 c 的最大汉明距离不超过 t 的所有向量。也就是说对于向量 t，如果 $d(c,u) \leq t$，则向量 u 属于汉明球 $B(c,t)$。通过下面的引理可以计算 $B(c,t)$ 中向量的个数。

引理： 一个 q 进制的 n 维空间汉明球 $B(c,t)$ 有

$$\binom{n}{0}+\binom{n}{1}(q-1)+\binom{n}{2}(q-1)^2+\cdots+\binom{n}{r}(q-1)^r$$

个元素。

证明： 首先我们计算与码字 c 的距离为 1 的向量个数，这些向量和码字 c 只有一个位置不同，因此有 n 种可能的位置和 $q-1$ 种情况使一个位置不同。现在计算与码字 c 的汉明距离为 m 的向量个数，与码字 c 有 m 个位置不同共有$\binom{n}{m}$种情况。对于 m 个位置的每一个，与码字 c 中对应的符号不同的情况又有 $q-1$ 种可能。因此有

$$\binom{n}{m}(q-1)^m$$

个向量与码字 c 的距离为 m，包括向量 c 自己，利用等式 $\binom{n}{0}=1$，我们得到结论：

$$\binom{n}{0}+\binom{n}{1}(q-1)+\binom{n}{2}(q-1)^2+\cdots+\binom{n}{r}(q-1)^r。$$ □

现在我们就可以描述**汉明边界（Hamming bound）**了，汉明边界也称作**球的填充边界（sphere packing bound）**。

定理： 假设 C 是一个 q 进制的（n，M，d）码，并且 $d\geqslant 2t+1$，那么

$$M\leqslant\frac{q^n}{\sum_{j=0}^{t}\binom{n}{j}(q-1)^j}。$$

证明：对于每一个码字 c，我们设置一个半径为 t 的汉明球，由于汉明距离 $d\geqslant 2t+1$，因此这些汉明球不会重叠。所有汉明球中包含的向量总数不超过 q^n。因此我们得到：

$$(\text{码字数量})\times(\text{每一个球中元素的个数})$$
$$=M\sum_{j=0}^{t}\binom{n}{j}(q-1)^j\leqslant q^n。$$

这就产生了关于 M 的不等式。 □

一个以等号满足汉明边界的最小距离 $d=2t+1$ 的(n,M,d)码被称为**理想码（perfect code）**。一个能纠正 t 个错误的理想码满足半径为 t 以码字为中心的汉明球所覆盖的 q 进制 n 元组。汉明码（16.5 节）和 Golay 码 $\mathcal{G}_{23}$（16.6 节）是理想码，其他理想码的例子有所有长度为 n 的元组构成的（n，q^n，1）码以及练习 15 中长度为奇数的二进制循环码。

对于理想码的研究很多，而且许多观点都很有趣。现在已经知道理想码的所有列表，包括前面的例子以及由 Golay 码构成的三进制［11，6，5］码。同时我们要告诫读者，虽然名字是理想码，但并不意味着理想码是最好的纠错码，比如 Reed-Solomon 码，虽然不是理想码，但在同样的条件下比理想码的纠错能力更强。

16.3.2　下边界条件

纠错码的一个核心问题是在给定长度和最小距离 d 的前提下找到最大的编码。这引出了下面的定义。

定义：假设字母表$\mathcal{A}$有 q 个元素，给定 n 和 d 并且 $d \leqslant n$，使一个（n，M，d）码存在的最大 M 表示为 $A_q(n,d)$。

我们通常可以找到至少一个（n，M，d）码：确定$\mathcal{A}$中的一个元素 a_0。假设 C 是所有向量（a，a，...，a，a_0，a_0，...，a_0）（包含 d 个 a 和 $n-d$ 个 a_0）的集合，其中 $a \in \mathcal{A}$。共有 q 个这样的向量，并且每个向量间的距离是 d，因此有一个（n，q，d）码。这给出了通常的下边界条件 A_q（n，d）$\geqslant q$。以后我们将会获得更好的边界条件。

很容易得到 $A_q(n,1)=q^n$：当一个码有最小距离 $d=1$ 时，我们能够选择所有 q 进制的 n 元组构成编码。另一个极端是 $A_q(n,n)=q$（练习 7）。

下面这些下边界条件被称为 **Gilbert-Varshamov 边界**，是在 1950 年发现的。

定理：给定 n 和 d 并且 $n \geqslant d$，存在一个 q 进制的（n，M，d）码，使得

$$M \geqslant \frac{q^n}{\sum_{j=0}^{d-1}\binom{n}{j}(q-1)^j}$$

即：

$$A_q(n,d) \geqslant \frac{q^n}{\sum_{j=0}^{d-1}\binom{n}{j}(q-1)^j}。$$

证明：从向量 c_1 开始，去掉$\mathcal{A}^n$（这里的$\mathcal{A}$是有 q 个符号的字母表）中以 c_1 为中心、半径为 $d-1$ 的汉明球中的所有向量。然后从剩余的向量中选择另外一个向量 c_2，由于所有与向量 c_1 距离不大于 $d-1$ 的向量都去掉了，所以 d（c_2，c_1）$\geqslant d$。现在去掉以 c_2 为中心、半径为 $d-1$ 的汉明球中的所有向量，并且从剩余向量中选 c_3。对于 c_3 我们不能得到 d（c_3，c_1）$\leqslant d-1$ 或 d（c_3，c_2）$\leqslant d-1$。因此对于 $i=1$，2，有 $d(c_3,c_i) \geqslant d$。对于 c_4，c_5，...，重复以上方法，直到没有满足的向量为止。

一个选中的向量最多从空间中去掉

$$\sum_{j=0}^{d-1}\binom{n}{j}(q-1)^j$$

个向量。如果我们选择 M 个向量 c_1，...，c_m，结合前面的引理，那么最多去掉

$$M\sum_{j=1}^{d-1}\binom{n}{j}(q-1)^j$$

个向量。我们能够继续直到所有 q^n 个向量被去掉。也就是说，能够至少继续到

$$M\sum_{j=1}^{d-1}\binom{n}{j}(q-1)^j \geqslant q^n。$$

因此存在一个关于 M 的编码 {c_1，...，c_M} 满足前面的不等式。

由于 $A_q(n,d)$ 是最大的 M，因此它也满足这个不等式。

还有一个小技术需要介绍一下，我们实际上已经建立了一个（n，M，e）码，并且 $e \geqslant d$。然而，通过修改 c_2 中的元素可以使 $d(c_2,c_1)=d$，剩下的向量将被上面的过程选择，这产生了一个最小距离是 d 的编码。 □

如果想通过一个噪声信道传送 n 比特码字，并且错误的概率是 p，那么当 n 很大时我们希望错误的个数大概是 pn。因此，需要一个 $d>2pn$ 的（n，M，d）码。于是对于给定的 $x>0$，我们需要考虑 $d/n \approx x>0$。下边界是如何影响 M 和编码率的呢？

下面进行说明。选择 q 和 x 满足 $0 < x < 1 - 1/q$。**Gilbert-Varshamov 边界的渐近线说明**有一个 q 进制的 (n, M, d) 码，当 $n \to \infty$，$d/n \to x$ 时编码率达到极限 $\geq H_q(x)$，其中

$$H_q(x) = 1 - x\log_q(q-1) + x\log_q(x) + (1-x)\log_q(1-x)。$$

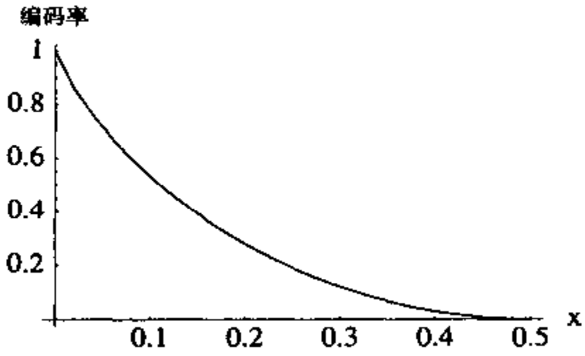


图 16.2 H_2 (x) 的曲线

图 16.2 是 H_2 (x) 的图像。当然我们希望编码有较高的纠错能力（即高的 x）和高的编码率（等于 k/n）。渐近的结果表明，在图像上无限接近或在曲线之上的位置，存在纠错能力和编码效率都很好的编码。

存在某编码序列的编码率严格大于 $H_q(x)$（对某个 x 和 q）的结论，是 Tsfasman，Vladut 和 Zink 使用代数几何中的 Goppa 码在 1982 年证明的。

16.3.3 例子

考虑长度为 3 的二进制循环码 C 的两个向量 (0, 0, 0) 和 (1, 1, 1)，这是一个 (3, 2, 3) 码。Singleton 边界条件指出，由于 $2 = M \leq 2$，所以 C 是一个 MDS 码。汉明边界指出：

$$2 = M \leq \frac{2^3}{\binom{3}{0} + \binom{3}{1}} = 2,$$

因此 C 是理想码。Gilbert-Varshamov 边界条件指出，存在一个二进制的 $(3, M, 3)$ 码，使得

$$M \geq \frac{2^3}{\binom{3}{0} + \binom{3}{1} + \binom{3}{2}} = \frac{8}{7},$$

这意味着 $M \geq 2$。

对于 [7, 4] 汉明码有 $M = 16$，$d = 3$，因此它是一个 (7, 16, 3) 码。Singleton 边界条件指出 $16 = M \leq 2^4$，因此是 MDS 码。汉明边界条件指出

$$16 = M \leq \frac{2^7}{\binom{7}{0} + \binom{7}{1}} = 16,$$

因此是理想码。而 Gilbert-Varshamov 边界条件指出，存在一个 $(7, M, 3)$ 码使得

$$M \geqslant \frac{2^7}{\binom{7}{0}+\binom{7}{1}+\binom{7}{2}} = \frac{128}{29} \approx 4.4,$$

因此汉明码比这个下边界条件好得多。有高效的纠错算法并且超出 Gilbert-Varshamov 边界条件的码目前比较少。

16.1 节中的阿达玛码是一个二进制（32，64，16）码。Singleton 边界条件指出 64 = $M \leqslant 2^{17}$，因此在这个例子中并不明显。汉明边界指出

$$64 = M \leqslant \frac{2^{32}}{\sum_{j=0}^{7}\binom{32}{j}} \approx 951.3。$$

而 Gilbert-Varshamov 边界条件指出，存在一个（32，M，16）码使得

$$M \geqslant \frac{2^{32}}{\sum_{j=0}^{15}\binom{32}{j}} \approx 2.3。$$

16.4 线 性 码

当用移动电话和一个朋友交谈时，你的声音在传输之前被转换成数字信号并进行纠错编码。你的朋友收到这些数据后，必须通过对纠错码解码来纠正传输中的错误。只有通过了解码，这些数据才能变成代表你噪音的声音。

在这种应用中，对一个数据包解码的时延是至关重要的。如果解码需要几秒钟的时间，那么时延将变得很严重并且会使交谈变得困难。

因此高效地解码是一个至关重要的问题。为了快速地解码，在编码中采用具有某种结构的编码要比采用$\mathcal{A}^n$的随机子集好得多。这是我们研究线性编码的主要原因，在本章的剩余部分我们将集中讨论线性编码。

在本书中字母表$\mathcal{A}$代表有限域 **F**。对于有限域的介绍，参见 3.10 节。大多数情况下读者可以假设有限域 **F** 即为集合 $\mathbf{Z}_2 = \{0,1\}$ =（整数 mod 2），在这种情况下我们使用二维向量。另外一个有限域的例子是 $\mathbf{Z}_p$ =（整数 mod 一个素数 p）。其他有限域的例子可以参见 3.10 节。在这里要特别指出，在特殊情况下 **F** 必须是有限域 $GF(q)$；但是这里所说的有限域的定义要简单些。由于使用任意的有限域，在等式中可以用 “=” 代替 “≡”。如果取有限域 **F** 为 $\mathbf{Z}_2$，只需要将 **F** 域中所有的元素模 2 取余。

F 中的元素组成的 n 维向量集合表示为 $\mathbf{F}^n$。n 维向量集合构成了 **F** 域上的向量空间，**F** 域上向量空间的子空间 S 对于线性变换是封闭的，也就是说对于 S 内的向量 s_1，s_2和 **F** 域上的 a_1，a_2，满足 $a_1 s_1 + a_2 s_2 \in S$。比如 $a_1 = a_2 = 0$，那么 $(0, 0, \ldots, 0) \in S$。

定义：有限域 **F** 上长度为 n 的 k 维线性编码是 n 维向量空间 $\mathbf{F}^n$的子空间。这样一个编码称为 [**n**, **k**] 码。如果码的最小距离是 d，那么这种编码称为 [**n**, **k**, **d**] 码。

当 $\mathbf{F} = \mathbf{Z}_2$时，定义可以简化，长度为 n 的 k 维二进制编码是 2^k进制 n 元组（码字）的集合，该元组满足任何两个码字之和总是一个码字。

我们遇到的大多数编码都是线性编码。例如二进制循环码 $\{(0,0,0),(1,1,1)\}$ 是向量空间 $\mathbf{Z}_2^3$ 的一维子空间。16.1 节例 2 中的奇偶校验码是一个长度为 8、维数为 7 的线性编码。它由长度为 8 的二进制向量组成，每个码字的元素之和的模 2 值为零。我们很容易得到构成一个子空间的对应向量集合。向量

$$(1,0,0,0,0,0,0,1),(0,1,0,0,0,0,0,1),\ldots,(0,0,0,0,0,0,1,1)$$

构成了这个向量空间的基础。由于有 7 个基本的向量，这个子空间是 7 维的。

16.1 节例 4 中的 [7，4] 汉明码是一个长度为 7 的 4 维线性码，每一个码字都涉及矩阵 G 中 4 行的线性变换，因此这 4 行生成了这个编码，并且是线性无关的，它们构成了基。

16.1 节例 5 中的 ISBN 码不是线性码。它由有限域 $\mathbf{Z}_{11}$ 上的元素组成的 10 维向量的集合组成。由于每个码字的前 9 个元素不能出现 X，因此它对线性变换并不是封闭的。

假设 C 是域 $\mathbf{F}$ 上的 k 维线性编码，如果 $\mathbf{F}$ 有 q 个元素，那么 C 有 q^k 个元素，下面解释一下原因。C 存在 k 个元素的一个基，称作 $v_1\cdots v_k$，C 中每一个元素都能用下面的形式惟一表示：$a_1v_1+\cdots+a_kv_k$，其中 $a_1,\ldots,a_k\in\mathbf{F}$。对于每个 a_i 有 q 个选择，而 a_i 的数目是 k，这意味着 C 有 q^k 个元素。因此一个 $[n,k,d]$ 线性编码也是 16.2 节所说的 (n,q^k,d) 码。

对于一个任意的、可能是非线性的编码，计算它的最小距离可能需要对每一对码字计算 $d(u,v)$。而对于一个线性编码，计算最小距离要简单的多。定义向量 u 的**汉明权（Hamming weight）** $wt(u)$ 为 u 中非零位的个数，如果用 0 表示向量 $(0,0,\ldots,0)$，那么 u 的汉明权等于 $d(u,0)$。

定理：假设 C 是一个线性码，那么最小距离 $d(C)$ 等于所有非零编码向量的汉明权的最小值，即 $d(C)=\min\{wt(u)\mid 0\neq u\in C\}$。

证明：由于 $wt(u)=d(u,0)$ 是两个码字的距离，因此对任意的码字 u，有 $wt(u)\geqslant d(C)$，它表明有一个码字的权重与 $d(C)$ 相等。注意对于任意两个向量 v，w，有 $d(v,w)=wt(v-w)$。这是因为 $v-w$ 的元素是非零的，并且当且仅当 v 和 w 在某个元素不同时，可以在 $wt(v-w)$ 中计算 $v-w$。选择不同的码字 v 和 w 使得 $d(v,w)=d(C)$，那么 $wt(v-w)=d(C)$，因此非零码的最小汉明权等于最小距离 $d(C)$。 □

为了构造一个线性 $[n,k]$ 码，我们必须构造一个 n 维向量空间 $\mathbf{F}^n$ 上的 k 维子空间。最简单的办法就是选择 k 个线性无关的向量，由它们生成 k 维子空间。可以选择一个由 $\mathbf{F}$ 域上元素组成的 $k\times n$ 的秩为 k **生成矩阵** G。当 v 遍历 $\mathbf{F}^k$ 中的所有行向量时，由向量集 vG 就得到子空间。

为了构造一个线性码，我们经常将 G 写成 $G=[I_k,P]$ 的形式，其中 I_k 是 $k\times k$ 的矩阵，P 是 $k\times(n-k)$ 的矩阵。矩阵 G 的行是长度为 n 的所有向量空间的 k 维子空间的基，这个子空间就是线性编码 C。换言之，每个码字都可以由矩阵 G 的行通过线性变换得到，如果使用矩阵 $G=[I_k,P]$ 构造一个码，前 k 列决定码字，后 $n-k$ 列提供冗余。

对于 16.1 节例 1 中编码的后半部分有：

$$G=\begin{pmatrix}1&0&1&0&1&0\\0&1&0&1&0&1\end{pmatrix}。$$

码字 101010 和 010101 作为矩阵的两行存在，而码字 111111 是这两行之和。这是一个 [6，2] 码。

例 2 中的编码有矩阵：

$$G=\begin{pmatrix}1&0&0&0&0&0&0&1\\0&1&0&0&0&0&0&1\\0&0&1&0&0&0&0&1\\0&0&0&1&0&0&0&1\\0&0&0&0&1&0&0&1\\0&0&0&0&0&1&0&1\\0&0&0&0&0&0&1&1\end{pmatrix}。$$

例如，码字 11001001 是矩阵 G 的第一行、第二行和第五行的模 2 和，因此可由（1，1，0，0，1，0，0）乘以 G 得到。这是一个［8，7］码。

在例 4 中编码的描述给出了矩阵 G，由它的名字就可以知道它是一个［7，4］码。

如前面所述，我们可以先构造秩为 k 的 $k\times n$ 矩阵。矩阵的行构成一个［n，k］码，而且行和列的操作可以将这个矩阵变换成所使用的矩阵 G，因此通常不使用这个一般的形式。前面所说的用矩阵 $G=[I_k,P]$ 描述一种编码是通常使用的方法。在这个例子中前 k 位是**信息位**，后 $n-k$ 位是**校验位**。

假设对于编码 C 有生成矩阵 $G=[I_k,P]$，令

$$H=[-P^T,I_{n-k}],$$

其中 P^T 是 P 的转置矩阵。在 16.1 节例 4 中矩阵 H 用来纠错，在例 2 中 $H=$［1，1，1，1，1，1，1，1］。注意，在这个例子中一个二进制串 v 当且仅当它的非零元素的个数是偶数时，或者说只有当 v 与 H 的点积为零时 v 才是一个码字。可以用公式表示为 $vH^T=0$，其中 H^T 是 H 的转置矩阵。

更一般的情况，假如有一个线性编码 $C\subset\mathbf{F}^n$。如果对于向量 v 满足 $v\in\mathbf{F}^n$，并且 v 是线性编码 C 中的码字，则当且仅当 $vH^T=0$ 时矩阵 H 称为**奇偶校验矩阵（parity check matrix）**。下面是一个有用的结论。

定理：如果 $G=[I_k,P]$ 是编码 C 的生成矩阵，那么 $H=[-P^T,I_{n-k}]$ 是编码 C 的奇偶校验矩阵。

证明：考虑矩阵 G 的第 i 行，其形式是：

$$v_i=(0,\ldots,1,\ldots,0,p_{i,1},\ldots,p_{i,n-k}),$$

其中 1 在第 i 个位置。v_i 是编码 C 的一个向量。H^T 的第 j 列为向量

$$(-p_{1,j},\cdots,-p_{n-k,j},0,\cdots,1,\cdots,0),$$

其中 1 在第 $n-k+j$ 个位置。为了获得 v_iH^T 的第 j 个元素，将上面两个向量点积得到：

$$1\cdot(-p_{i,j})+p_{i,j}\cdot 1=0。$$

因此 H^T 消灭 G 中的每一行 v_i。由于 C 中每个元素都是 G 中行向量的和，因此对于所有的 $v\in C$ 有 $vH^T=0$。

复习一下有关的线性代数知识：秩为 r 的 $m\times n$ 矩阵的左边零空间的维数是 $n-r$。由于 H^T 包含子矩阵 I_{n-k}，矩阵 I_{n-k} 的秩为 $n-k$，因此 H^T 左边的零空间的维数为 k。刚才我们证明了 C 是包含在这个零空间中的，因此 C 的维数一定与这个零空间相同，故 C 的维数也是 k。

现在我们有检查错误的方法了：如果 v 是传输中收到的信息并且 $vH^T\neq 0$，那么一定有错误发生。如果 $vH^T=0$，我们并不能确定没有错误，但能确定 v 是一个码字。由于没有错误发

生的可能性比发生了很多错误使一个码字变为另一个码字的可能性要大，所以我们认为没有错误发生。

我们也可以利用奇偶校验矩阵使解码的过程变得简单，首先看一个例子。

例：假设 C 是一个二进制的线性编码，C 的生成矩阵为

$$G=\begin{pmatrix}1&0&1&1\\0&1&1&0\end{pmatrix}。$$

可以根据下面的过程建立长度为 4 的二进制向量表。首先从（0，0，0，0，0）开始列出编码 C 中的 4 个元素作为第一行，然后从剩余的 12 个向量中选择权值最小的一个（可能不止一个）与第一行相加得到第二行，再从剩余 8 个向量中选择权值最小的一个与第一行相加得到第三行，最后从剩余 4 个向量中选择权值最小的一个与第一行相加得到第四行。得到下表：

(0,0,0,0)	(1,0,1,1)	(0,1,1,0)	(1,1,0,1)
(1,0,0,0)	(0,0,1,1)	(1,1,1,0)	(0,1,0,1)
(0,1,0,0)	(1,1,1,1)	(0,0,1,0)	(1,0,0,1)
(0,0,0,1)	(1,0,1,0)	(0,1,1,1)	(1,1,0,0)

这个表能够用作解码表。当接收到一个向量时，在这个表中查找它，通过把这个向量变成它对应列的第一个元素来解码，而改正的错误就是该向量对应行的第一个元素。例如，假设我们接收到了向量（0，1，0，1），它是第二行的最后一个元素，解密为（1，1，0，1），即去掉了错误（1，0，0，0）。在这个小例子中和最小邻域解码方法不完全相同，比如（0，0，1，0）解码为（0，1，1，0），而它还有一个距离相同的邻域（0，0，0，0），这个问题是由于码的最小距离是 2 引起的，所以一般的纠错是不可能的。如果使用能够纠正最多 t 个错误的编码，那么只要与一个码字的距离不超过 t 的所有向量都能用这个过程正确解码。

对于一个大的例子，在这样一个表里查找向量是很麻烦的。事实上写这样一个表更加困难（这是我们采用一个小例子的原因），于是奇偶校验矩阵 H 出现了。

一行中第一个向量 v 称为**陪集首（coset leader）**。令 r 是这个表中与 v 同一行的任意向量，根据这个表的构造方法，存在码字 c 使得 $r=v+c$，由奇偶校验矩阵的定义有 $cH^T=0$。因此，

$$rH^T = vH^T + cH^T = vH^T,$$

向量 $S(r)=rH^T$ 称为向量 r 的**伴随式（syndrome）**，那么同一行的两个向量有相同的伴随式。我们用下面较小的表取代前面的表。

陪集首	伴随式
(0,0,0,0)	(0,0)
(1,0,0,0)	(1,1)
(0,1,0,0)	(1,0)
(0,0,0,1)	(0,1)

利用这个表可按下面的方式进行解码。对于收到的一个向量 r，计算它的伴随式 $S(r)=rH^T$，然后在列表中找到这个伴随式并用 r 减去对应的陪集首，这样可以完成对 r 的解码。例如，$r=(0,1,0,1)$，那么

$$S(r) = (0,1,0,1)\begin{pmatrix}1&1\\1&0\\1&0\\0&1\end{pmatrix} = (1,1)。$$

这是第二行的伴随式，从 r 中减去陪集首（1, 0, 0, 0）得到码字（1, 1, 0, 1）。 ■

现在我们考虑更一般的情况。通过例子中的方法可以得到两个定义。

定义：假设 C 是一个线性编码，u 是一个 n 维的向量，集合 $u+C$ 由下式给出：

$$u + C = \{u + c \mid c \in C\}$$

称集合 $u + C$ 为 C 的**陪集**。

很容易发现如果 $v \in u + C$，那么集合 $v + C$ 与集合 $u + C$ 是相同的（练习 9）。

定义：在陪集中有最小汉明权的向量称为**陪集首**。

向量 u 的伴随式定义为 $S(u) = uH^T$。下面的引理可用来方便地确定陪集。

引理：当且仅当两个向量 u 和 v 含有相同的伴随式时，这两个向量属于同一个陪集。

证明：两个向量 u 和 v 属于同一个陪集，当且仅当这两个向量属于编码 C 中不同的向量时成立，即 $u - v \in C$。当且仅当 $(u-v)H^T = 0$ 时 $u - v \in C$ 成立，$(u-v)H^T = 0$ 又等价于 $S(u) = uH^T = vH^T = S(v)$。 □

通过建立伴随式查询表可以实现解码，伴随式查询表包含陪集首和它对应的伴随式。利用一个伴随式查询表，我们可以通过下面的步骤解码：

1. 对于一个收到的向量 r，计算它的伴随式 $S(r) = rH^T$。
2. 查找 $S(r)$ 对应的陪集首，记为 c_0。
3. 将 r 解码为 $r - c_0$。

对于收到的一个向量，伴随式解码比寻找距离最近的码字解码需要的步骤少得多，然而，对于大量的编码，伴随式解码实施起来效率仍然不高。一般而言在线性码中寻找最近的邻域是很困难的，它被称为 NP-完全问题。然而对于某些特殊类型的编码，高效率的解码是可能的。我们会在下面的章节里看到一些例子。

对偶码

向量空间 $\mathbf{F}^n$ 有一个点积，定义如下：

$$(a_1, \ldots, a_n) \cdot (b_0, \ldots, b_n) = a_0 b_0 + \cdots + a_n b_n。$$

例如，如果 $\mathbf{F} = \mathbf{Z}_2$，那么

$$(0,1,0,1,1,1) \cdot (0,1,0,1,1,1) = 0,$$

我们发现，点积与实数相乘不同，一个非零的向量与其自身的点积总为零。虽然点积不能告诉我们向量的长度，但点积是一个重要的概念。

如果 C 是一个线性 $[n, k]$ 码，定义**对偶码**（**dual code**）如下：

$$C^{\perp} = \{u \in \mathbf{F}^n \mid 对于所有的 c \in C 有 u \cdot c = 0\}。$$

定理：如果 C 是一个线性 $[n, k]$ 码，其生成矩阵为 $G = [I_k, p]$，那么 $C^{\perp}$ 是一个生成矩阵为 $H = [-P^T, I_{n-k}]$ 的线性 $[n, n-k]$ 码，而且 G 是 $C^{\perp}$ 的奇偶校验矩阵。

证明：由于 C 中每一个元素都是 G 中行的线性变换，当且仅当 $uG^T = 0$ 时向量 u 在 $C^{\perp}$ 上，这意味着 $C^{\perp}$ 是 G^T 左边的零空间。而且，G 是 $C^{\perp}$ 的奇偶校验矩阵。因此 G 的秩为 k，同样 G^T 的秩也为 k。G^T 左边的零空间维数是 $n-k$，因此 $C^{\perp}$ 的维数也是 $n-k$。由于 H 是 C 的奇偶校验矩阵，而且 G 的行都是编码 C 中的元素，因此有 $GH^T = 0$。对这个等式求转置，由于转置可以掉换位置（$(AB)^T = B^T A^T$），可以得到 $HG^T = 0$。这意味着 H 中的行向量在 G^T 左边的零空间内；因此也在对偶码 $C^{\perp}$ 中。由于 H 的秩是 $n-k$，因此 H 行向量的维数是 $n-k$，这和 $C^{\perp}$ 的维数相等。由于 H 的行向量可以生成 $C^{\perp}$，因此 H 是 $C^{\perp}$ 的生成多项式。 □

一个码的对偶码是它自己，即 $C=C^{\perp}$ 称 C 为**自对偶码**（**self-dual**），16.6 节中的 $\mathcal{G}_{24}$ 码就是自对偶码的一个重要例子。

举例：假设 $C=\{(0,0,0),(1,1,1)\}$ 是一个二进制循环码。因此对于所有的 u 有 $u\cdot(0,0,0)=0$，当且仅当 $u\cdot(1,1,1)=0$ 时 u 是 $C^{\perp}$ 中的元素。这意味着 $C^{\perp}$ 是一个奇偶校验码：当且仅当 $a_1+a_2+a_3=0$ 时 $(a_1,a_2,a_3)\in C^{\perp}$。■

举例：C 是二进制编码，并且生成矩阵为

$$G=\begin{pmatrix}1&0&0&1\\0&1&1&0\end{pmatrix}。$$

由前面的定理得到 $C^{\perp}$ 的生成矩阵为

$$H=\begin{pmatrix}0&1&1&0\\1&0&0&1\end{pmatrix}。$$

H 是由 G 中的行向量交换位置得到的，故 G 和 H 的行向量产生相同的子空间。因此 $C=C^{\perp}$，即 C 为自对偶码。■

16.5 汉　明　码

汉明码是很重要的一类单一的纠错码，它能够很容易地检错、纠错，最初它使用在长途电话的差错控制中。二进制的汉明码有下面的参数：

1. 码长：$n=2^m-1$。
2. 维数：$k=2^m-m-1$。
3. 最小距离：$d=3$。

描述汉明码的最简单的方法是通过它的奇偶校验矩阵。对于长度 $n=2^m-1$ 的二进制汉明码，首先构造一个 $m\times n$ 的矩阵，其列都是长度为 m 的二进制数组。比如 [7，4] 二进制汉明码，我们选择 $m=3$，于是 $n=7$，$k=4$，并且先构造矩阵：

$$\begin{pmatrix}1&0&1&0&1&0&1\\0&1&1&0&0&1&1\\0&0&0&1&1&1&1\end{pmatrix}。$$

为了获得一个码的奇偶校验矩阵的正规形式，我们把相应的列移到矩阵的后边，使矩阵的后面变为 $m\times m$ 的一个单位矩阵，其他列的顺序无所谓。结果就是汉明 $[n,k]$ 码的奇偶校验矩阵 H。在这个例子中，我们将第四、第二、第一列移到矩阵的后面，得到下面的矩阵

$$H=\begin{pmatrix}1&1&0&1&1&0&0\\1&0&1&1&0&1&0\\0&1&1&1&0&0&1\end{pmatrix},$$

这就是例 3 的矩阵 H。

可以很方便地由奇偶校验矩阵 H 计算生成矩阵 G。由于汉明码是单一的纠错码，用伴随式解码很容易。特别地，错误向量 e 允许有至多为 1 的权重，因此它是零，或者除了在第 j 个位置有一个 1 以外其余全部为零。

最多纠正一位错误的汉明码定义如下：

1. 对接收到的向量 y 计算伴随式 $s=yH^T$。如果 $s=0$，那么没有错误，返回 y 后退出。

2. 否则，确定伴随式 s 的转置与 H 中第 j 列相等。

3. 变换接收到的向量 y 的第 j 比特，输出解码。

只要接收到的向量中最多有一个错误，那么结果将是发送的码字。

举例：[15，11] 二进制汉明码有如下的奇偶校验矩阵：

$$\begin{pmatrix} 0&0&0&0&1&1&1&1&1&1&1&1&0&0&0\\ 1&1&1&0&0&0&0&1&1&1&1&0&1&0&0\\ 0&1&1&1&0&1&1&0&0&1&1&0&0&1&0\\ 1&0&1&1&1&0&1&0&1&0&1&0&0&0&1 \end{pmatrix}。$$

假设接收的向量为

$$y=(0,0,0,0,1,0,0,0,0,0,1,1,0,0,1),$$

计算伴随式 $s=yH^T$ 的结果是 $s=(1,1,1,1)$。由于 s 是 H 中第 11 列的转置，因此改变 y 中第 11 个比特得到码字：

$$(1,0,0,1,0,0,0,0,1,0,0,0,0,0,0),$$

由于前 11 个比特给出的是信息，因此原始信息是

$$(1,0,0,1,0,0,0,0,1,0,0)。$$ ■

16.6 Golay 码

在著名的二进制编码中有两个 Golay 码，$\mathcal{G}_{23}$ 和 $\mathcal{G}_{24}$。[24，12，8] 扩展的 Golay 码 $\mathcal{G}_{24}$ 第一次使用是在 1978～1981 年 *Voyager I* 号和 *Voyager II* 号航天飞机向地球传送木星、土星的彩色照片时提供纠错码。Golay 码 $\mathcal{G}_{23}$（非扩展的）是一个 [23，12，7] 码，接近于 $\mathcal{G}_{23}$ 码。我们先构造 $\mathcal{G}_{24}$，然后再修改得到 $\mathcal{G}_{23}$。还有其他一些构造 Golay 码的方法，参见 [MacWilliams-Sloane]。

$\mathcal{G}_{24}$ 的生成矩阵是一个 12×24 的矩阵 $G=$

$$\begin{pmatrix}
1&0&0&0&0&0&0&0&0&0&0&0&1&1&1&0&1&1&1&0&0&0&1&0\\
0&1&0&0&0&0&0&0&0&0&0&0&1&0&1&1&0&1&1&1&0&0&0&1\\
0&0&1&0&0&0&0&0&0&0&0&0&1&1&0&1&1&0&1&1&1&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0&1&0&1&0&1&1&0&1&1&1&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&1&0&1&1&0&1&1&1&0\\
0&0&0&0&0&1&0&0&0&0&0&0&1&0&0&0&1&0&1&1&0&1&1&1\\
0&0&0&0&0&0&1&0&0&0&0&0&1&1&0&0&0&1&0&1&1&0&1&1\\
0&0&0&0&0&0&0&1&0&0&0&0&1&1&1&0&0&0&1&0&1&1&0&1\\
0&0&0&0&0&0&0&0&1&0&0&0&1&1&1&1&0&0&0&1&0&1&1&0\\
0&0&0&0&0&0&0&0&0&1&0&0&1&0&1&1&1&0&0&0&1&0&1&1\\
0&0&0&0&0&0&0&0&0&0&1&0&1&1&0&1&1&1&0&0&0&1&0&1\\
0&0&0&0&0&0&0&0&0&0&0&1&0&1&1&1&1&1&1&1&1&1&1&1
\end{pmatrix},$$

G 中的所有元素都是模 2 的整数值。G 中的前 12 列是一个 12×12 的单位矩阵。后 11 列由下面的方式得到，平方 mod 11 的值为 0，1，3，4，5，9（例如 $4^2 \equiv 3$，$7^2 \equiv 5$）。考虑向量 $(x_0, \ldots, x_{10}) = (1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0)$，在位置 0，1，3，4，5，9 上为 1。这给出了 G 中第一行的后 11 个元素，除了最后一行，其他行都是由第一行的元素循环位移变换得到的（注意所有的元素都是模 2 的整数，而不是模 11 的。平方模 11 只是用来决定哪个位置存放 1）。第 13 列和第 12 行是相同的，只是增加了 k 和 d，并且使码有一些很好的属性。$\mathcal{G}_{24}$的基本属性将在下面给出。

定理：$\mathcal{G}_{24}$是一个自对偶 [24，12，8] 二进制码。$\mathcal{G}_{24}$中所有向量的权重都是 4 的倍数。

证明：G 中行的长度为 24。G 中包含 12×12 的单位矩阵，则 G 中的 12 行是线性无关的，因此$\mathcal{G}_{24}$的维数是 12，可见是一个 [24，12，d] 码。下面的事情就是证明 $d=8$，接着再证明$\mathcal{G}_{24}$是一个自对偶码并且它的码字的权重为 0（模 4）。

当然，可以用计算机列出$\mathcal{G}_{24}$所有的 $2^{12}=4096$ 个元素以及它们的权重，随即就可以证明这个定理。然而，我们更愿意给出一个理论上的证明。

假设 r_1是 G 中的第一行，并且令 $r \neq r_1$是其他 11 行中的任意一行。很容易发现 r 和 r_1都有 4 个 1 位置相同，并且 4 个 1 的每一个都与其他向量 0 的位置相对应。在 $r+r_1$中 4 个相同的 1 抵消了，因此 $r+r_1$中有 8 个 1，即权重为 8，并且 r，r_i的点积为 $r \cdot r_i = 1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 = 4 \equiv 0 \pmod 2$。

现在我们假设 u 和 v 是 G 中除最后一行之外的任意两个不同的行，v 中的前 12 个元素和后 11 个元素是 u 中对应部分循环移位得到的，同时也可由第一行的对应部分循环移位得到。由于移位操作不改变向量的权重也不改变点积值，因此将前面对 $r+r_1$，$r \cdot r_1$的计算过程应用于 u 和 v。于是有：

1. $wt\ (u+v)=8$
2. $u \cdot v \equiv 0 \pmod 2$

简单的检查发现，上面两个等式对于 G 中最后一行也成立，因此上面的结论对于 G 中任何不同的 u，v 都成立。同时由于 G 中有偶数个 1，因此等式（2）在 $u=v$ 时也成立。

现在假设 c_1，c_2是$\mathcal{G}_{24}$中的任意两个元素，那么 c_1，c_2是 G 中行的线性变换，因此 $c_1 \cdot c_2$是许多 $u \cdot v$ 的线性变换，其中 u，v 是 G 中的行向量。由于每个 $u \cdot v \equiv 0 \pmod 2$，于是 $r_1 \cdot r_2 \equiv 0 \pmod 2$，这意味着 $C \subseteq C^{\perp}$。因为 C 是 24 维空间中的 12 维子空间，$C^{\perp}$的维数是 $24-12=12$，可见 C 和 $C^{\perp}$有相同的维数，并且二者互相包含。所以 $C=C^{\perp}$，即 C 是自对偶码。

G 中所有行的权重都是 4 的倍数，下面的引理将说明$\mathcal{G}_{24}$中每个元素都是 4 的倍数。

引理：假设 v_1，v_2是相同长度的二进制向量，那么

$$wt(v_1+v_2) = wt(v_1) + wt(v_2) - 2[v_1 \cdot v_2],$$

表达式 $[v_1 \cdot v_2]$ 意味着将点积视为一个普通的整数，而不模 2（比如 $[(1, 0, 1, 1) \cdot (1, 1, 1, 1)]=3$，而不是 1）。

证明：只有当 v_1，v_2中一个元素是 1 而另一个对应元素为 0 时，v_1+v_2中才会出现一个 1。当两个向量都有一个 1 时，v_1+v_2中对应元素是 0 模 2。注意，$wt\ (v_1)\ +wt\ (v_2)$ 包含了 v_1和 v_2中所有的 1，包括 v_1+v_2中相互抵消的部分，而 $[v_i \cdot v_2]$ 的值恰好是两个向量中相

同的1，因此在 v_1，v_2 中各有 $[v_1 \cdot v_2]$ 个1包含在 $wt(v_1)+wt(v_2)$ 中，而不包含在 $wt(v_1+v_2)$ 中。综合上述内容就得到了引理中的等式。 □

现在我们回到定理的证明中。考虑 $\mathcal{G}_{24}$ 中的一个向量 g，它可以写成 $g \equiv u_1+\cdots+u_k \pmod 2$，其中 $u_1,\ldots,u_k$ 是 G 中不同的行。我们将通过归纳 k 证明 $wt(g) \equiv 0 \pmod 4$。先来看 G，我们知道 G 中所有的行权重都是4的倍数，因此 $k=1$ 时成立。通过归纳，假设所有能用 G 中 $k-1$ 行元素表达的向量都有权重 $\equiv 0 \pmod 4$，确切地说，$u=u_1+\cdots+u_{k-1}$ 的权重是4的倍数。由引理得到：

$$wt(g)=wt(u+u_k)=wt(u)+wt(u_k)-2[u \cdot u_k] \equiv 0+0-2[u \cdot u_k] \pmod 4。$$

由于我们已证明 $[u \cdot u_k] \equiv 0 \pmod 2$，因此 $2[u \cdot u_k] \equiv 0 \pmod 4$。这样就证明了当 g 是 k 行之和时 $wt(g) \equiv 0 \pmod 4$。通过归纳可以得出，G 中所有行之和有权重 $\equiv 0 \pmod 4$。这就证明了 $\mathcal{G}_{24}$ 的所有权重都是4的倍数。

最后我们证明 $\mathcal{G}_{24}$ 中最小的权重是8，它对 G 中所有的行都成立，但还需要证明对于 G 中行的和也成立。由于码字的权重是4的倍数，因此，我们需要证明没有权重为4的码字，因为权重至少是8。事实上8是最小的，因为 G 中第一行的权重就是8。

我们需要下面的引理。

引理：由 G 中后12列构成的 12×12 矩阵 B 的行是模2线性无关的。由 G 中前11行的后11个元素构成的 11×11 矩阵 A 的行是模2线性相关的，惟一的线性相关性是 A 中所有11行的和模2等于0。

证明：由于 $\mathcal{G}_{24}$ 是自对偶的，因此 G 中任意两行的点积都是零。这意味着 $GG^T=0$。由于 $G=[I|B]$（即 I 后面跟着矩阵 B），上式可以写为：

$$I^2+BB^T=0,$$

这表明 $B^{-1}=B^T$（工作在模2的方式下，所以求逆的符号消失）。即 B 是可逆的，因此 B 中的行是线性无关的。

A 中行向量的和模2为0，因此这是一个相关的关系。假设 $v_1=(1,\ldots,1)^T$ 是一个11维的列向量，那么 $Av_1=0$，这同样说明了 A 中所有行的和为0。假设 v_2 是一个非零的11维的列向量并且 $Av_2=0$，在 v_1，v_2 列向量的前面加上一个0，使其扩展为12维的列向量 v_1'，v_2'。假设 r_{12} 是 B 中最后一行，那么

$$Bv_i'=(0,\ldots,0,r_{12} \cdot v_i')^T,$$

这个等式由 $Av_i=0$ 得到。注意矩阵乘以一个向量是由矩阵中的行与这个向量点积得到的。

由于 B 是可逆的并且 $v_i' \neq 0$，有 $Bv_i' \neq 0$，因此 $r_1 \cdot v_i' \neq 0$，由于我们考虑的是模2的情况，则点积值一定是1。因此

$$B(v_1'+v_2')=(0,\ldots,0,r_1 \cdot v_1'+r_1 \cdot v_2')^T=(0,\ldots,0,1+1)^T=0,$$

由于 B 是可逆的，必须有 $v_1'+v_2'=0$，因此 $v_1'=v_2'$（此时工作的是模2方式）。不考虑 v_1'，v_2' 最上面的元素，可以得到 $v_2=(1,\ldots,1)$。因此在 A 的零空间中惟一的非零向量就是 v_1。一个矩阵零空间的向量给出了矩阵中行向量的线性相关性，这样我们得到结论：矩阵 A 中惟一的线性关系就是所有行的和为0。这就证明了引理。证毕。 □

假设 g 是 $\mathcal{G}_{24}$ 中的码字，由于 G 中的前12列构成了一个单位矩阵，因此，如果 g 是 G 中第二行、第三行、第七行的和，那么 g 将在第二、第三、第七个位置有元素1。这样我们

发现如果 g 是 k 行的和，那么 $wt(g)\geqslant k$。假设 $wt(g)=4$，那么 g 最多是 G 中 4 行的和，很明显 g 不可能是 G 中单独一行元素，因为每一行的权重至少为 8。如果 $g=r_1+r_2+r_3$ 是 G 中三行的和，那么有两种可能。

（1）首先假设 g 不包括 G 中最后一行。由于使用了第 13 列的 3 个 1，因此有一个 1 出现在 g 的第 13 个位置处。前 12 列对 g 又提供了 3 个 1（r_1，r_2，r_3 每行一个）。因为 $wt(g)=4$，这样我们已经找到了 g 中的 4 个 1，那么 g 中剩余 11 个元素都为 0。由前面的引理，矩阵 A 的三行相加不可能为 0，因此这种情况不可能。

（2）假设 g 包括 G 中最后一行，$g=r_1+r_2+r_3$，其中 $r_3=G$ 中最后一行，那么 g 中后 11 个元素将由 A 中（来自 r_1 和 r_2）两行之和加上 r_3 中的向量（1，1，...，1）组成。前面我们已证明了 G 中两行相加的权重是 8，前 13 列对权重的贡献是 2，那么后 11 列的贡献是 6。加上向量（1，1，...，1）将所有的 1 变为 0，所有的 0 变为 1。因此 g 中后 11 个元素的权重是 5。可见 $wt(g)=4$ 是不可能的，因此后一种情况也不可能发生。

最后，如果 g 是 G 中 4 行的和，那么 g 中前 12 个元素的和有 4 个 1。因此 g 中后 12 个元素全是 0。由引理知道 B 中 4 个元素之和不可能是 0，两者相矛盾。这样我们就证明了没有一个码字的权重是 4。

既然权重是 4 的倍数，那么最小的可能是 8。正如我们前面指出的那样，存在权重为 8 的码字，于是我们证明了 $\mathcal{G}_{24}$ 的最小权重是 8，因此 $\mathcal{G}_{24}$ 是一个［24，12，8］码。这就完成了定理的证明。□

（非扩展的）Golay 码 $\mathcal{G}_{23}$ 是由 $\mathcal{G}_{24}$ 去掉每个码字的最后一项得到的。

定理：$\mathcal{G}_{23}$ 是一个线性［23，12，7］码。

证明：显然每一个码字长度是 23。同样，很容易看出 $\mathcal{G}_{23}$ 中的向量集合也可以看作是对加法封闭的（如果 v_1，v_2 是长度为 24 的向量，那么 v_1+v_2 的前 23 个元素可以看作是 v_1，v_2 前 23 个元素相加得到的），并且 $\mathcal{G}_{23}$ 构成了一个二进制向量空间。$\mathcal{G}_{23}$ 的生成矩阵 G' 可以由 $\mathcal{G}_{24}$ 的生成向量 G 去掉最后一列得到。由于 G' 包含了 12×12 的单位矩阵，因此 G' 的行是线性无关的，并且生成了一个 12 维的向量空间。如果 g' 是 $\mathcal{G}_{23}$ 中的一个码字，那么 g' 可以通过去掉 $\mathcal{G}_{24}$ 中的某个码字 g 的一项得到。如果 $g'\neq0$，那么 $g\neq0$，因此 $wt(g)\geqslant8$。由于 g' 比 g 少一项，因此有 $wt(g')\geqslant7$。定理证明完毕。□

$\mathcal{G}_{24}$ 码的解码

假设一个消息用 $\mathcal{G}_{24}$ 编码，并且收到包含最多 3 个错误的消息。下面我们介绍一种纠错的方法。

令 G 是 $\mathcal{G}_{24}$ 的一个 12×24 的生成矩阵。将 G 写成下面的形式

$$G=[I,B]=(c_1,\cdots,c_{24}),$$

其中 I 是一个 12×12 的单位矩阵，B 包含了 G 中的后 12 列，并且 c_1，...，c_{24} 是列向量。这里 c_1，...，c_{12} 是 12 维空间的标准基元素。记为

$$B^T=(b_1,...,b_{12}),$$

其中 b_1，...，b_{12} 是列向量。这意味着 b_1^T，...，b_{12}^T 是 B 中的行。

假设收到的消息 $r=c+e$，其中 c 是 $\mathcal{G}_{24}$ 中的码字，并且

$$e=(e_1,...,e_{24})$$

是错误向量。我们假设 $wt(e) \leqslant 3$。

算法如下所示，证明将在后面给出。

1. 令 $s = rG^T$ 是伴随式。

2. 比较行向量 s，sB，$s + c_j^T$（其中 $13 \leqslant j \leqslant 24$）和 $sB + b_j^T$（其中 $1 \leqslant j \leqslant 12$）。

3. 如果 $wt(s) \leqslant 3$，那么 s 中非零的元素对应 c 中非零的元素。

4. 如果 $wt(sB) \leqslant 3$，只有当 e 中第 $k+12$ 个元素非零时，sB 中第 k 个位置也为非零。

5. 如果对于 $13 \leqslant j \leqslant 24$ 有 $wt(s + c_j^T) \leqslant 2$，那么 $e_j = 1$，并且 $s + c_j^T$ 中的非零位置与错误向量 e 中非零的位置对应。

6. 如果对于 $1 \leqslant j \leqslant 12$ 有 $wt(sB + b_j^T) \leqslant 2$，那么 $e_j = 1$。如果对于 $s + b_j^T$ 在位置 k 处有一个非零的元素（最多有两个这样的 k），那么 $e_{12+k} = 1$。

举例：待发送的原始消息为

$$m = (1,1,0,0,0,0,0,0,0,0,1,0),$$

码字计算为：

$$mG = (1,1,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,1,0,1,0,1,1,0)$$

并将码字发送给我们。假设接收到的消息为：

$$r = (1,1,0,1,0,0,0,0,0,0,1,0,1,0,0,0,0,1,0,0,0,0,1,0),$$

计算显示

$$s = (0,1,1,1,1,0,1,1,0,0,1,0)$$

并且

$$sB = (1,0,1,0,1,1,0,0,1,0,0,0)。$$

所有这些向量的权重都大于3，因此可以计算 $s + c_j^T$（$13 \leqslant j \leqslant 24$）和 $sB + b_j^T$（$1 \leqslant j \leqslant 12$）。我们发现

$$sB + b_4^T = (0,0,0,0,0,0,0,1,0,1,0,0),$$

这意味着在位置 4 处（对应选择 b_4）、位置 20（$=12+8$）处、22（$=12+10$）处（对应 $sB + b_4^T$ 中位置 8 和 10 的非零元素）各有一个错误。我们可以计算

$$c = r + (0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0)$$
$$= (1,1,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,1,0,1,0,1,1,0)。$$

而且，由于 G 是一个系统的形式，我们可以由前 12 个元素恢复原始信息：

$$m = (1,1,0,0,0,0,0,0,0,0,1,0)。$$

■

这样我们就证明了这个算法，并且显示如果 $wt(e) \leqslant 3$，至少有前面的一个错误发生。

由于 $\mathcal{G}_{24}$ 是自对偶的，G 中一行与任何码字 c 的点积是 0，这意味着 $cG^T = 0$。在例子中，有 $r = c + e$，因此

$$s = rG^T = cG^T + eG^T = eG^T = e_1 c_1^T + \cdots + e_{24} c_{24}^T。$$

最后的等式表明向量 $e = (e_1, \ldots, e_{24})$ 乘以 G^T 等于 e_1 乘以 G^T 中的第一行 c_1^T 加上 e_2 乘以 G^T 中的第二行，等等。而且

$$sB = eG^TB = e\begin{bmatrix} I \\ B^T \end{bmatrix} B = e\begin{bmatrix} B \\ I \end{bmatrix},$$

因为 $B^T = B^{-1}$（由前面的引理得到）。我们有

$$\begin{bmatrix} B \\ I \end{bmatrix} = [B^T, I]^T = (b_1, \ldots, b_{12}, c_1, \ldots, c_{12})。$$

因此，

$$sB = e(b_1, \ldots, b_{12}, c_1, \ldots, c_{12})^T = e_1 b_1^T + \cdots + e_{24} c_{12}^T。$$

如果 $wt(e) \leqslant 3$，那么 $wt((e_1, \ldots, e_{12})) \leqslant 1$ 或者 $wt((e_{13}, \ldots, e_{24})) \leqslant 1$，否则 e 中将会有过多的非零元素。因此我们可以考虑下面的 4 种情况，

1. $wt((e_1, \ldots, e_{12})) = 0$。那么

$$sB = e_{13} c_1^T + \cdots + e_{24} c_{12}^T = (e_{13}, \ldots, e_{24}),$$

因此，$wt(sB) \leqslant 3$，并且可以在算法的步骤 4 中确定错误。

2. $wt((e_1, \ldots, e_{12})) = 1$。那么在 $1 \leqslant j \leqslant 12$ 中只有一个 j 使得 $e_j = 1$，因此

$$sB = b_j^T + e_{13} c_1^T + \cdots + e_{24} c_{12}^T。$$

因此

$$sB + b_j^T = e_{13} c_1^T + \cdots + e_{24} c_{12}^T = (e_{13}, \ldots, e_{24})。$$

向量（e_{13}，…，e_{24}）最多有两个非零元素，因此符合算法的第 6 步。j 由 sB 惟一确定。假设对于某些 $k \neq j$，有 $wt(sB + b_k^T) \leqslant 2$。那么

$$wt(b_k^T + b_j^T) = wt(sB + b_k^T + sB + b_j^T)$$
$$\leqslant wt(sB + b_k^T) + wt(sB + b_j^T) \leqslant 2 + 2 = 4$$

（见练习 6）。然而，在证明 $\mathcal{G}_{24}$ 码的定理时，知道 G 中不同两行之和的权重为 8，因此可知 B 中不同两行之和的权重为 6，有 $wt(b_k^T + b_j^T) = 6$。这个矛盾说明 b_k 不可能存在，所以 b_j 是惟一的。

3. $wt((e_{13}, \ldots, e_{24})) = 0$。在这种情况下

$$s = e_1 c_1^T + \cdots + e_{12} c_{12}^T = (e_1, \ldots, e_{12})。$$

我们已知 $wt(s) \leqslant 3$，因此满足算法的第 3 步。

4. $wt((e_{13}, \ldots, e_{24})) = 1$。在这种情况下，对于 $13 \leqslant j \leqslant 24$ 中的某个 j 有 $e_j = 1$，因此

$$s = e_1 c_1^T + \cdots + e_{12} c_{12}^T + c_j^T,$$

并且可以得到：$s + c_j^T = e_1 c_1^T + \cdots + e_{12} c_{12}^T = (e_1, \ldots, e_{12})$。

在（e_1，…，e_{12}）中最多有两个非零的元素，因此满足算法的第 5 步。

和情况（2）中一样，c_j 的选择也是由 s 惟一确定的。

在每一种情况中，我们得到了一个向量，记为 e'，其中最多有 3 个非零的元素。为了纠正这些错误，我们把 e' 与收到的向量 r 相加（或减；此时工作在模 2 的方式下）得到 $c' = r + e'$。怎样才能知道这就是发送的向量呢？对于选择的 e'，我们有

$$e' G^T = s,$$

因此

$$c' G^T = r G^T + e' G^T = s + s = 0。$$

由于 $\mathcal{G}_{24}$ 是自对偶的，G 是 $\mathcal{G}_{24}$ 的一个奇偶校验矩阵。由于 $c' G^T = 0$，我们得到结论 c' 是一个码字。通过纠正 r 中的最多 3 个错误得到 c'。由于我们假定最多有 3 个错误，并且由于 $\mathcal{G}_{24}$ 的最小权重是 8，c' 一定是正确的解码。因此，正如前面所申明的那样，这个算法能够正确地解码。

前面的算法需要许多步骤，且需要计算26个向量的权重。为什么不看看有3个错误的各种可能，以及纠正哪一种能够产生一个码字？对至多有3个错误的各种位置共有$\binom{24}{0}+\binom{24}{1}+\binom{24}{2}+\binom{24}{3}=2325$种可能，这样可以由计算机来完成。但是，前面所述的解码算法还是要快一些。

16.7 循 环 码

循环码是非常重要的一类码。在下面的两节里，我们将遇到这些码中最有用的两个例子。在这一节我们将描述一般的框架。

一个码C如果满足

$$(c_1,c_2,\ldots,c_n)\in C\text{ 那么 }(c_n,c_1,c_2,\ldots,c_{n-1})\in C。$$

则称码C是**循环码（cyclic）**。例如，如果（1，1，0，1）是一个循环码，那么（1，1，1，0）也是循环码。多次运用这个定义会发现（0，1，1，1）和（1，0，1，1）也是码字，因此码字的所有循环位移也是码字。让一个码满足这个条件可能看起来很困难。毕竟，对于给定的一个码字，它的循环移位仍然是码字，这看起来似乎不相关。关键在于循环码有许多结构，这使循环码比较容易学习。在BCH码的实例中（参见16.8节），这个结构产生了一个高效的解码算法。

让我们以一个例子开始，考虑二进制矩阵

$$G=\begin{pmatrix}1&0&1&1&1&0&0\\0&1&0&1&1&1&0\\0&0&1&0&1&1&1\end{pmatrix},$$

G中的行产生了一个7维二进制空间的3维子空间。事实上，对第一行循环移位得到所有的非零码字：

G = {（0，0，0，0，0，0，0），（1，0，1，1，1，0，0），（0，1，0，1，1，1，0），（0，0，1，0，1，1，1），（1，0，0，1，0，1，1，），（1，1，0，0，1，0，1），（1，1，1，0，0，1，0）（0，1，1，1，0，0，1）}。

很明显最小权重是4，因此我们有循环［7，3，4］码。

下面我们提供一个代数方法得到这个码。用$\mathbf{Z}_2[X]$表示它是系数模2的X的多项式，用$\mathbf{Z}_2[X]/(X^7-1)$表示多项式模(X^7-1)。更详细的描述参见3.10节。现在我们只需要知道模(X^7-1)意味着所处理的多项式阶数小于7。如果有一个多项式的阶数大于或等于7，那就除以(X^7-1)取余项。

假设$g(X)=1+X^2+X^3+X^4$。考虑与阶数不大于2的$f(X)$点积。

$$g(X)f(X)=a_0+a_1X+\cdots+a_6X^6$$

将这个乘积写成一个向量$(a_0,\ldots,a_6)$。例如$g(X)\cdot 1$产生（1，0，1，1，1，0，0），这是G中的第一行。同样$g(X)X$得到G的第二行，$g(X)X^2$得到G的第三行。而且$g(X)(1+X^2)$得到（1，0，0，1，0，1，1），这是第一行和第三行的和。这样可以得到该码的所有码字。

通过考虑点积 $g(X)f(X)$ 我们得到这个码，其中 $\deg(f) \leqslant 2$。也可以通过任意阶的 $f(X)$ 得到同样的码，只要工作在模 (X^7-1) 方式下。注意到 $g(X)(X^3+X^2+1)=X^7-1 \pmod 2$。将 X^3+X^2+1 从 $f(X)$ 中分离：

$$f(X)=(X^3+X^2+1)q(X)+f_1(X),$$

利用 $\deg(f_1) \leqslant 2$，有

$$\begin{aligned} g(X)f(X) &= g(X)(X^3+X^2+1)q(X)+g(X)f_1(X) \\ &= (X^7-1)q(X)+g(X)f_1(X) \equiv g(X)f_1(X) \bmod (X^7-1)。\end{aligned}$$

可见 $g(X)f_1(X)$ 给出了与 $g(X)f(X)$ 相同的码字。因此我们可以限制只考虑多项式的阶数最多为 2。

为什么这个码是循环码呢？首先看向量 $g(X)$，向量 $g(X)X$，$g(X)X^2$ 是分别由 $g(X)$ 循环移位 1 位、2 位得到的。如果向量 $g(X)$ 乘以 X^3 呢？得到一个阶数为 7 的多项式，因此除以 X^7-1 并保留余式：

$$g(X)X^3 = X^3+X^5+X^6+X^7=(X^7-1)(1)+(1+X^3+X^5+X^6),$$

余式产生了向量 (1, 0, 0, 1, 0, 1, 1)。这是由向量 $g(X)$ 循环移动 3 位得到的。

对于 $j=4, 5, 6$ 的类似计算表明 $g(X)X^j$ 对应的向量由 $g(X)$ 对应的向量循环移动 j 位得到。事实上这只是表面的现象。如果 $q(X)=a_0+a_1X+\cdots+a_6X^6$ 是一个多项式，那么

$$\begin{aligned} q(X)X &= a_0X+a_1X^2+\cdots+a_6X^7 \\ &= a_6(X^7-1)+a_6+a_0X+a_1X^2+\cdots+a_5X^6, \end{aligned}$$

余式是 $a_6+a_0X+a_1X^2+\cdots+a_5X^6$，对应向量为 $(a_6, a_0, \ldots, a_5)$。因此乘以 X 并模 X^7-1 对应向量循环位移 1 位。重复这个过程 j 次可知乘以 X^j 结果是对应向量循环位移 j 位。

现在来描述一般的情况。假设 $\mathbf{F}$ 是一个有限域，对于一个有限域的处理，参见 3.10 节。这里我们认为 $\mathbf{F}$ 是整数模 p 的剩余类，其中 p 是一个素数。比如你可以认为 $\mathbf{F}=\mathbf{Z}_2=\{0, 1\}$ 是整数模 2。假设 $\mathbf{F}[X]$ 表示以 $\mathbf{F}$ 中的元素为系数的 X 的多项式，选择一个正整数 n，我们考虑多项式 $\mathbf{F}[X]/(X^n-1)$，它描述了 $\mathbf{F}[X]$ 中的元素模 (X^n-1)。这意味着处理的多项式阶数小于 n。当所计算的多项式的阶数 $\geqslant n$ 时，将其除以 X^n-1 取余式。假设 $g(X)$ 是 $\mathbf{F}(X)$ 上的多项式，考虑多项式的集合

$$m(X)=g(X)f(X) \quad \bmod (X^n-1),$$

其中 $f(X)$ 遍历 $\mathbf{F}(X)$ 中的所有多项式（由于高阶的多项式能够通过 $\bmod X^n-1$ 降阶，故我们只考虑 $f(x)$ 阶数小于 n 的）。将 $m(X)$ 写为：

$$m(X)=a_0+a_1X+\cdots+a_{n-1}X^{n-1}。$$

其中的系数给出了一个 n 维向量 $(a_0, \ldots, a_{n-1})$。所有这些系数的集合构成了 n 维空间 $\mathbf{F}^n$ 的一个子空间 C。那么 C 是一个码。

如果 $m(X)=g(X)f(X) \bmod (X^n-1)$ 是任意一个这样的多项式，并且 $s(X)$ 是另外一个多项式，那么 $m(X)s(X)=g(X)f(X)s(X) \bmod (X^n-1)$ 是多项式 $f(X)s(X)$ 与 $g(X)$ 的乘积。这样它产生了码 C 中的一个元素。特殊情况下，乘以 X 并且模 X^n-1 对应于由最初的码字循环位移一位得到的码字。因此 C 是一个循环码。

下面的定理给出了循环码的一般描述。

定理：假设 C 是有限域 $\mathbf{F}$ 上长度为 n 的循环码。对于每一个码字 $(a_0, \ldots, a_{n-1}) \in$

C，与 $\mathbf{F}[X]$ 上的多项式 $a_0+a_1X+...+a_{n-1}X^{n-1}$ 相关联。在通过这种方法得到的所有非零多项式中，假设 $g(X)$ 有最小的阶数，通过分离最高位系数我们可以假设 $g(X)$ 中最高非零位系数是 1，多项式 $g(X)$ 被称为 C 的**生成多项式（generating polynomial）**。那么

1. $g(X)$ 由 C 惟一确定。
2. $g(X)$ 是 X^n-1 的因子。
3. C 是多项式 $g(X)f(X)$ 的系数，其中 $\deg(f)\leq n-1-\deg(g)$。
4. 记 $X^n-1=g(X)h(X)$，当且仅当 $h(X)m(X)\equiv 0 \bmod (X^n-1)$ 时 $m(X)\in \mathbf{F}[X]/(X^n-1)$ 对应于 C 中的一个元素。

证明：

(1) 如果 $g_1(X)$ 是另外一个这样的多项式，那么 $g(X)$ 和 $g_1(X)$ 有相同的阶数，并且最高非零位的系数是 1。由于 C 对于减法是封闭的，因此 $g(X)-g_1(X)$ 有较低的阶数并且对应一个码字。由于 $g(X)$ 在所有码字对应的多项式中有最小的阶数，因此 $g(X)-g_1(X)$ 必定是 0，这意味着 $g(X)=g_1(X)$。因此 $g(X)$ 是惟一的。

(2) 从 X^n-1 分离出 $g(X)$：

$$X^n-1=g(X)h(X)+r(X)$$

对于某些多项式 $g(X)$ 和 $r(X)$，其中 $\deg(r)<\deg(g)$。这意味着

$$-r(X)\equiv g(X)h(X) \bmod (X^n-1),$$

正如前面所说的，$g(X)$ 乘以 X 的幂对应于 $g(X)$ 相应的码字循环移位。由于 C 是循环码，多项式 $g(X)X^j \bmod (X^n-1)$ 对于 $j=0, 1, 2, ...$ 对应的是码字，称之为 $c_0, c_1, c_2, ...$，记 $h(X)=b_0+b_1X+\cdots+b_kX^k$。那么 $g(X)h(x)$ 对应于线性变换

$$b_0c_0+b_1c_1+\cdots+b_kc_k。$$

由于每个 b_i 都在 $\mathbf{F}$ 内，每个 c_i 都在 C 中，我们有了一个 C 中元素的线性变换。但是 C 是一个 n 维向量空间 $\mathbf{F}^n$ 的向量子空间，因此这个线性变换就在 C 中，这意味着 $r(X)$，即 $g(X)h(X) \bmod (X^n-1)$，对应一个码字。但是 $\deg(r)<\deg(g)$，而 g 是 C 中非零码字对应的最小阶数的多项式，有 $r(X)=0$。因此 $X^n-1=g(X)h(X)$，故 $g(X)$ 是 X^n-1 的因子。

(3) 假设 $m(X)$ 对应 C 中的一个元素。从 $m(X)$ 中分离出 $g(X)$：

$$m(X)=g(X)f(X)+r_1(X),$$

其中 $\deg(r_1(X))<\deg(g(X))$。和前而一样，$g(X)f(X) \bmod (X^n-1)$ 对应于一个码字，假设 $m(X)$ 也对应于一个码字，那么 $m(X)-g(X)h(X) \bmod (X^n-1)$ 对应与这些码字不同的一个码字。但是这个多项式为 $r_1(X)=r_1(X) \bmod (X^n-1)$。和前面的证明一样，这个多项式的阶数小于 $\deg(g(X))$，这样 $r_1(X)=0$，因此 $m(X)=g(X)f(X)$，由于 $\deg(m)\leq n-1$，必须有 $\deg(f)\leq n-1-\deg(g)$。在（2）中已证明，由于 C 是一个循环码，任何 $g(X)f(X)$ 形式的多项式产生一个码字。因此这些多项式产生的恰是 C 中的元素。

(4) 记 $X^n-1=g(X)h(X)$，这在（2）中已证明。假设 $m(X)$ 对应 C 中的一个元素。那么由（3）可得 $m(X)=g(X)f(X)$，因此

$$h(x)m(X)=h(X)g(X)f(X)=(X^n-1)f(X)\equiv 0 \bmod (X^n-1)。$$

相反，假设 $m(X)$ 是一个满足 $h(x)m(X)\equiv 0 \bmod (X^n-1)$ 的多项式。对于某些多项式 $q(X)$，记 $h(x)m(X)=(X^n-1)q(X)=h(X)g(X)q(X)$，将它除以 $h(X)$ 得到 $m(X)=g(X)q(X)$，$m(X)$ 是 $g(X)$ 的一个乘积，因此对应着一个码字。这样就完成了定理的证明。 □

和定理中一样，假设 $g(X)=a_0+a_1X+\cdots+a_{k-1}X^{k-1}+X^k$。由定理的第（3）部分知道：$C$ 中的每个元素对应 $g(X)f(X)$ 形式的一个多项式，其中 $\deg(f(X))\leqslant n-1-k$。这意味着每个这样的 $f(X)$ 是单项式 $1, X, X^2, \ldots, X^{n-1-k}$ 的线性变换。接着得到：C 中的码字是多项式 $g(X), g(X)X, g(X)X^2, \ldots, g(X)X^{n-1-k}$ 对应码字的线性变换。但是这些向量是

$$(a_0,\ldots,a_k,0,0,\ldots),(0,a_0,\ldots,a_k,0,\ldots),\ldots,(0,\ldots,0,a_0,\ldots,a_k)。$$

因此，C 的一个生成矩阵可以由下式给出：

$$G=\begin{pmatrix} a_0 & a_1 & \cdots & a_k & 0 & 0 & \cdots \\ 0 & a_0 & a_1 & \cdots & a_k & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & a_0 & a_1 & \cdots & a_k \end{pmatrix}。$$

我们可以利用定理得到码 C 的一个奇偶校验矩阵。和定理中一样，假设 $h(X)=b_0+b_1X+\cdots+b_lX^l$（其中 $l=n-k$）。我们将证明 $k\times n$ 矩阵

$$H=\begin{pmatrix} b_l & b_{l-1} & \cdots & b_0 & 0 & 0 & \cdots \\ 0 & b_l & b_{l-1} & \cdots & b_0 & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & b_l & b_{l-1} & \cdots & b_0 \end{pmatrix}$$

是编码 C 的奇偶校验矩阵。注意 $h(X)$ 的系数顺序是颠倒的。H 是码 C 的奇偶校验矩阵意味着当且仅当 $c\in C$ 时 $Hc^T=0$。

定理：H 是 C 的奇偶校验矩阵。

证明：首先我们注意到 $g(X)$ 以 1 作为它的最高非零位的系数，并且 $g(X)h(X)=X^n-1$，可见 $h(X)$ 的最高非零位的系数 b_l 一定是 1。因此 H 是行阶梯的形式，并且它的行是线性无关的。由于 H 有 k 行，因此它的秩也是 k。所以 H 右边的零空间是 $n-k$ 维的。

假设 $m(X)=c_0+c_1X+\cdots+c_{n-1}X^{n-1}$。我们由定理的第（4）部分知道当且仅当 $h(X)m(X)\equiv 0 \bmod (X^n-1)$ 时 $(c_0,c_1,\ldots,c_{n-1})\in C$。

选择满足 $l\leqslant j\leqslant n-1$ 的 j，并且考察点积 $h(X)\ m(X)$ 中 X^j 的系数。它等于：

$$b_0c_j+b_1c_{j-1}+\cdots+b_{l-1}c_{j-l+1}+b_lc_{j-l}。$$

这里还有一个知识点需要介绍：既然我们考虑 $h(X)m(X) \bmod (X^n-1)$，就需要考虑 X^{n+j} 的影响（由于 $X^{n+j}\equiv X^nX^j\equiv 1\cdot X^j$，单项式 X^{n+j} 可以被降次为 X^j）。然而，点积 $h(X)m(X)$ 的最高阶数在模 X^n-1 之前是 $c_{n-1}X^{l+n-1}$。因为 $l\leqslant j$，我们有 $l+n-1<j+n$。所以，不必担心项 X^{n+j} 的影响。

当我们乘以 H 次 $(c_0, c_1, \ldots, c_{n-1})^T$ 时，得到一个向量，它的第一个元素是

$$b_lc_0+b_{l-1}c_1+\cdots+b_0c_l。$$

更一般的情况，第 i 个元素是（其中 $1\leqslant i\leqslant k$）

$$b_lc_{i-1}+b_{l-1}c_i+\cdots+b_0c_{l+i-1},$$

这是 $h(X)m(X) \bmod (X^n-1)$ 中 X^{l+i-1} 的系数。

如果 $(c_0, c_1, ..., c_{n-1})$ 是 C 中的系数，而 $h(X)m(X) \equiv 0 \bmod (X^n-1)$，因此所有这些系数是 0。因此乘以 H 次 $(c_0, c_1, ..., c_{n-1})^T$ 得到的是 0 向量，C 中向量的转置存在于 H 右边的零空间内。由于 C 和零空间都是 k 维的，我们可以得到等式。这就证明了当且仅当 $Hc^T=0$ 时 $c \in C$，这意味着 H 是 C 的奇偶校验矩阵。 □

举例：在这一节开始的例子里，有 $n=7$ 并且 $g(X)=X^4+X^3+X^2+1$。我们有 $g(X)(X^3+X^2+1)=X^7-1$，因此 $h(X)=X^3+X^2+1$。奇偶校验矩阵是

$$H=\begin{pmatrix}1&1&0&1&0&0&0\\0&1&1&0&1&0&0\\0&0&1&1&0&1&0\\0&0&0&1&1&0&1\end{pmatrix}。$$ ■

这个奇偶校验矩阵给出了一种检错的方法，但是纠错对于一般的循环码来说通常是很困难的。下一节将介绍一类有好的解码算法的纠错码。

16.8 BCH 码

BCH 码是一类循环码，它大约在 1959 年由博斯（R. C. Bose）和雷-查德胡里（D. K. Ray-Chaudhuri）两人以及霍昆格姆（A. Hocquenghem）个人分别提出。BCH 码重要的原因之一是它存在好的解码算法（例如，参见［Gallager］或［Wicker］）。BCH 码使用在卫星上。特殊的 BCH 码称为 Reed-Solomon 码（见 16.9 节），它有广泛的应用。

在描述 BCH 码之前，我们需要知道有限域的知识。假设 $\mathbf{F}$ 是有 q 个元素的有限域。从 3.10 节中我们知道 $q=p^m$ 是一个素数 p 的幂。假设 n 是一个不能被 p 整除的正整数，可以证明存在有限域 $\mathbf{F}'$ 包含有限域 $\mathbf{F}$，$\mathbf{F}'$ 包含了 1 的 n 次本原根 α。这意味着 $\alpha^n=1$，但是对于 $1\leq k<n$ 有 $\alpha^k\neq 1$。

例如，如果 $\mathbf{F}=\mathbf{Z}_2$，它是整数模 2 的，对 $n=3$，我们采用 $\mathbf{F}'=GF(4)$。3.10 节中描述的 $GF(4)$ 中元素 ω 是 1 的 3 次本原根。更一般的情况，当且仅当 n 能整除 $q'-1$ 时，即 $n\mid q'-1$ 时，在有限域 $\mathbf{F}'$ 中存在一个有 q' 个元素的 n 次本原根。

需要辅助域 $\mathbf{F}'$ 的原因是我们执行的许多运算需要在这个大的域中进行。下面当使用到 1 的 n 次本原根 α 时，我们将隐含假设：计算是在一个包含 α 的域 $\mathbf{F}'$ 中进行的。然而计算的结果将在较小的域 $\mathbf{F}$ 上给出编码的结果。

下面的结论经常称为 **BCH 边界条件**，给出了一个循环码最小权重的估计。

定理：设 C 是一个在有限域 $\mathbf{F}$ 上的 $[n, k, d]$ 循环码，其中 $\mathbf{F}$ 有 $q=p^m$ 个元素，假设 p 不能整除 n，设 $g(X)$ 是 C 的生成多项式。设 α 是一个 1 的 n 次本原根，并且对于某些整数 ℓ 和 δ 有

$$g(\alpha^{\ell})=g(\alpha^{\ell+1})=\cdots=g(\alpha^{\ell+\delta})=0,$$

那么 $d\geq\delta+2$。

证明：假设 $(c_0, c_1, ..., c_{n-1})\in C$ 的权重 w 满足 $1\leq w<\delta+2$。我们需要得到一个矛盾的结果。假设 $m(X)=c_0+c_1X+\cdots+c_{n-1}X^{n-1}$。已知 $m(X)$ 是 $g(X)$ 的倍数，因此

$$m(\alpha^{\ell}) = m(\alpha^{\ell+1}) = \cdots = m(\alpha^{\ell+\delta}) = 0。$$

假设 c_{i_1}，c_{i_2}，…，c_{i_w}是 m（X）的非零系数，因此

$$m(X) = c_{i_1}X^{i_1} + c_{i_2}X^{i_2} + \cdots + c_{i_w}X^{i_w}。$$

在 $l \leqslant j \leqslant \ell + w - 1$（注意 $w - 1 \leqslant \delta$）上有 m（α^j）$=0$，这可以重写成：

$$\begin{pmatrix} \alpha^{\ell i_1} & \cdots & \alpha^{\ell i_w} \\ \alpha^{(\ell+1)i_1} & \cdots & \alpha^{(\ell+1)i_w} \\ \vdots & \ddots & \vdots \\ \alpha^{(\ell+w-1)i_1} & \cdots & \alpha^{(\ell+w-1)i_w} \end{pmatrix} \begin{pmatrix} c_{i_1} \\ c_{i_2} \\ \vdots \\ c_{i_w} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix},$$

我们认为矩阵的行列式是非零的。我们需要下面的 Vandermonde 决定因子的评估。关于它的证明能在所有的线性代数书中找到。

定理：

$$\det \begin{pmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_n \\ x_1^2 & x_2^2 & \cdots & x_n^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{n-1} & x_2^{n-1} & \cdots & x_n^{n-1} \end{pmatrix} = \prod_{1 \leqslant i < j \leqslant n} (x_j - x_i)$$

（乘积是所有满足 $1 \leqslant i < j \leqslant n$ 的整数对（i，j）成立）。特殊地，如果 x_1，...，x_n是两两不同的，那么决定因子是非零的。

在所给的矩阵中，我们能够从第一列中因数分解 $\alpha^{\ell i_1}$，从第二列中因数分解 $\alpha^{\ell i_2}$，等等，这样可以得到

$$\det \begin{pmatrix} \alpha^{\ell i_1} & \cdots & \alpha^{\ell i_w} \\ \alpha^{(\ell+1)i_1} & \cdots & \alpha^{(\ell+1)i_w} \\ \vdots & \ddots & \vdots \\ \alpha^{(\ell+w-1)i_1} & \cdots & \alpha^{(\ell+w-1)i_w} \end{pmatrix}$$

$$= \alpha^{\ell i_1 + \cdots + \ell i_w} \det \begin{pmatrix} 1 & \cdots & 1 \\ \alpha^{i_1} & \cdots & \alpha^{i_w} \\ \vdots & \ddots & \vdots \\ \alpha^{(w-1)i_1} & \cdots & \alpha^{(w-1)i_w} \end{pmatrix}。$$

由于 α^{i_1}，…，α^{i_w}是两两不同的，因此决定因子是非零的。为什么这些数是不同的呢？假设 $\alpha^{i_j} = \alpha^{i_k}$。我们可以假设 $i_j \leqslant i_k$，有 $0 \leqslant i_j \leqslant i_k < n$，因此 $0 \leqslant i_k - i_j < n$。注意到 $\alpha^{i_k - i_j} = 1$。由于 α 是 1 的 n 次本原根，对于 $1 \leqslant i < n$，$\alpha^i \neq 1$。因此 $i_k - i_j = 0$，故 $i_j = i_k$。这意味着 α^{i_1}，…，α^{i_w}，是两两不同的。

由于决定因子非零，因此矩阵是非奇的。这意味着（c_{i_1}，...，c_{i_w}）$=0$，与存在非零的 c_i 相矛盾，因此所有的非零码字有最小权重 $\delta + 2$。这就完成了定理的证明。 □

举例：假设 $\mathbf{F} = \mathbf{Z}_2$ = 模 2 整数，且 $n = 3$。令 g（X）$= X^2 + X + 1$，那么

$$C = \{(0,0,0),(1,1,1)\},$$

这是一个二进制循环码。假设 ω 是 1 的 3 次本原根，就像 3.10 节中 GF（4）的描述一样，

那么$g(\omega)=g(\omega^2)=0$。通过这个定理能够得到$\ell=1$，$\delta=1$。我们发现C的最小权重至少为3。在本例中，由于最小权重至少为3，因此边界条件是明显的。■

举例：假设$\mathbf{F}$是任何有限域，n是任何正整数。设$g(X)=X-1$，那么$g(1)=0$，可以得到$\ell=0$，$\delta=0$。我们可以得到结论：由$g(X)$产生的码的最小权重至少为2（准确地说定理有假设p不能整除n，但是在这个特殊的例子中$\ell=\delta=0$，这个假设不是必需的）。我们以前已讨论过这个码。如果$(c_0,\ldots,c_{n-1})$是一个向量，并且$m(X)=c_0+\cdots+c_{n-1}X^{n-1}$是伴随多项式，那么只有当$m(1)=0$时，$m(X)$是$X-1$的倍数，这意味着$c_0+\cdots+c_{n-1}=0$。因此，当且仅当一个向量的元素之和是零时，该向量是一个码字。当$\mathbf{F}=\mathbf{Z}_2$时它是一个奇偶校验码，并且对于其他的有限域它是奇偶校验码的一般形式。很容易发现它的最小权重是2：如果一个码字有非零的元素，那么它必定有另外的非零元素将其抵消，并且使所有元素的和为零。因此每个非零的码字至少有两个非零的元素，可见权重至少为2。向量$(1,-1,0,\ldots)$是一个码字并且权重为2，因此最小权重是2。■

举例：我们再次考虑16.7节中7位的二进制循环码。有$\mathbf{F}=\mathbf{Z}_2$，并且$g(X)=1+x^2+x^3+x^4$。我们可以因式分解$g(X)=(X-1)(X^3+X+1)$。设α是X^3+X+1的一个根，那么α是1的7次本原根（见练习18），并且我们在$GF(8)$上考虑。由于$\mathbf{Z}_2\subset GF(8)$，有$2=1+1=0$和$-1=1$，因此$\alpha^3=\alpha+1$。平方结果产生$\alpha^6=\alpha^2+2\alpha+1=\alpha^2+1$。因此$(\alpha^2)^3+(\alpha^2)+1=0$。这意味着$g(\alpha^2)=0$，因此

$$g(1)=g(\alpha)=g(\alpha^2)=0。$$

由定理有$\ell=1$和$\delta=2$，所以码的最小权重至少为4（实际上就是4）。■

为了定义BCH码，我们需要更多的符号，这里在有限域$\mathbf{F}$上构造一个长度为n的码，在$\mathbf{F}$上因式分解X^n-1成不可约的因子：

$$X^n-1=f_1(X)f_2(X)\cdots f_r(X),$$

其中$f_i(X)$是以$\mathbf{F}$中元素为系数的多项式，并且每个$f_i(X)$不能通过$\mathbf{F}$中的系数因式分解为较低阶数的多项式。我们可以假设$f_i(X)$最高的非零系数是1。假设α是1的n次本原根，那么$\alpha^0,\alpha^1,\alpha^2,\ldots,\alpha^{n-1}$是$X^n-1$的根。这意味着

$$X^n-1=(X-1)(X-\alpha)(X-\alpha^2)\cdots(X-\alpha^{n-1})。$$

可见每个$f_i(X)$是某些$(X-\alpha^j)$的乘积，并且每个α^j是多项式$f_i(X)$的一个根。对于每个j，设$q_j(X)$是使$f_i(\alpha^j)=0$的多项式$f_i(X)$，这样得到下列多项式$q_0(X)$，$q_1(X)$，…$q_{n-1}(X)$。当然，由于多项式$f_i(X)$有两个不同幂的根α^j，α^k对应$q_j(X)$，$q_k(X)$（见本节后面的例子），通常这些多项式并不是明显的。

对于某个整数k，一个**设计距离为d的BCH码**的生成多项式是

$$g(X)=q_{k+1}(X),q_{k+2}(X),\ldots,q_{k+d-1}(X)$$

的最小公倍数。

定理：一个设计距离为d的BCH码是一个最小权重至少为d的码。

证明：由于在$k+1\leqslant j\leqslant k+d-1$上$q_j(X)$可除尽$g(X)$，并且$q_j(\alpha^j)=0$，我们有

$$g(\alpha^{k+1})=g(\alpha^{k+2})=\cdots=g(\alpha^{k+d-1})=0,$$

BCH边界条件意味该码的最小权重至少为$d=\delta+2$。□

举例：假设$\mathbf{F}=\mathbf{Z}_2$，并且$n=7$，那么

$$X^7 - 1 = (X-1)(X^3+X^2+1)(X^3+X+1)。$$

设 α 是 X^3+X+1 的一个根，那么和前面的例子一样，α 是 1 的 n 次本原根。而且，在前面的例子中我们证明了 α^2也是 X^3+X+1 的一个根。事实上，我们准确地得到 X^3+X+1 根的平方也是它的一个根，因此我们有 $\alpha^4=(\alpha^2)^2$也是 X^3+X+1 的一个根（我们能够继续平方，但是 $\alpha^8=\alpha$，于是回到了出发点）。因此 α，α^2，α^4是 X^3+X+1 的根，则有

$$X^3+X+1=(X-\alpha)(X-\alpha^2)(X-\alpha^4)。$$

α 剩下的幂次必定是 X^3+X^2+1 的根，则有

$$X^3+X^2+1=(X-\alpha^3)(X-\alpha^5)(X-\alpha^6)。$$

因此，

$$q_0(X)=X-1, q_1(X)=q_2(X)=q_4(X)=X^3+X+1,$$
$$q_3(X)=q_5(X)=q_6(X)=X^3+X^2+1。$$

如果我们取 $k=-1$，$d=3$，那么

$$g(X)=\mathrm{lcm}(q_0(X),q_1(X))$$
$$=(X-1)(X^3+X+1)=X^4+X^3+X^2+1,$$

我们得到 16.7 节中讨论的 [7，3，4] 循环码。定理指出码的最小权重至少是 3。在这个例子中，我们能做一些改进。如果我们取 $k=-1$，$d=4$，那么我们有生成多项式 $g_1(X)$ 为

$$g_1(X)=\mathrm{lcm}(q_0(X),q_1(X),q_2(X))=g(X)。$$

这是因为 $q_2(X)=q_1(X)$，因此当包含 $q_2(X)$ 时最小公倍数不改变。定理告诉我们码的最小权重至少是 4。正如我们前面看到的那样，最小权重是 4。■

举例（续）：我们继续讨论上面的例子，但是取 $k=0$，$d=7$。那么

$$g(X)=\mathrm{lcm}(q_1(X),\ldots,q_6(X))=(X^3+X+1)(X^3+X^2+1)$$
$$=X^6+X^5+X^4+X^3+X^2+X+1。$$

我们得到有两个码字的循环码：

$$\{(0,0,0,0,0,0,0),(1,1,1,1,1,1,1)\},$$

定理指出最小权重至少是 7。事实上最小权重就是 7。■

举例：假设 $\mathbf{F}=\mathbf{Z}_5=\{0,1,2,3,4\}$ =模 5 整数，取 $n=4$，那么

$$X^4-1=(X-1)(X-2)(X-3)(X-4)$$

（在有限域 $\mathbf{Z}_5$中这是一个等式或同余式）。取 $\alpha=2$，有 $2^4=1$，但是对于 $1\le j<4$ 有 $2^j\ne 1$。因此，2 是 $\mathbf{Z}_5$域上 1 的 4 次本原根。我们有 $2^0=1$，$2^2=4$，$2^3=3$（这些是模 5 同余）。因此

$$q_0(X)=X-1, q_1(X)=X-2, q_2(X)=X-4, q_3(X)=X-3,$$

在定理中，取 $k=0$，$d=3$。那么

$$g(X)=\mathrm{lcm}(q_1(X),q_2(X))=(X-2)(X-4)$$
$$=X^2-6X+8=X^2+4X+3。$$

我们得到一个 $\mathbf{Z}_5$域上的 [4，2] 循环码，并且有生成矩阵

$$\begin{pmatrix}3&4&1&0\\0&3&4&1\end{pmatrix}。$$

定理指出最小权重至少是 3。由于矩阵的头一行是一个权重为 3 的码字，所以最小权重就是 3。这个码是一个 Reed-Solomon 码的例子，下一节将进行讨论。■

解 BCH 码

BCH 码有用的原因之一就是它有一些好的解码算法，其中最著名的是 Berlekamp 和 Massey 算法（见［Gallager］或［Wicker］）。下面我们不给出具体的算法，但为了给出解码所包含的思想，我们演示纠正一个设计距离 $d\geq 3$ 的 BCD 码中一个错误的方法。

设 C 是一个设计距离 $d\geq 3$ 的 BCH 码，那么 C 是一个长度为 n 的循环码，且生成多项式为 $g(X)$。存在一个 1 的 n 次本原根 α，对于某些整数 k 满足

$$g(\alpha^{k+1})=g(\alpha^{k+2})=0$$

设

$$H=\begin{pmatrix}1 & \alpha^{k+1} & \alpha^{2(k+1)} & \cdots & \alpha^{(n-1)(k+1)}\\ 1 & \alpha^{k+2} & \alpha^{2(k+2)} & \cdots & \alpha^{(n-1)(k+2)}\end{pmatrix}。$$

如果 $c=(c_0,\ ...,\ c_{n-1})$ 是一个码字，那么多项式 $m(X)=c_0+c_1X+\cdots+c_{n-1}X^{n-1}$ 是 $g(X)$ 的倍数，因此

$$m(\alpha^{k+1})=m(\alpha^{k+2})=0,$$

这可以写成含 H 的形式：

$$cH^T=(c_0,c_1,...,c_{n-1})\begin{pmatrix}1 & 1\\ \alpha^{k+1} & \alpha^{k+2}\\ \alpha^{2(k+1)} & \alpha^{2(k+2)}\\ \vdots & \vdots\\ \alpha^{(n-1)(k+1)} & \alpha^{(n-1)(k+2)}\end{pmatrix}=0。$$

由于在 H 的零空间中可能有非码字存在，因此 H 对于 C 不是一个必要的奇偶校验矩阵。然而，正如我们所看到的，H 能够纠正一个错误。

假设收到了向量 $r=c+e$，其中 c 是一个码字，并且 $e=(e_0,\ ...,\ e_{n-1})$ 是一个错误向量。假定 e 中最多有一个非零元素。

这里有纠正一个错误的算法。

1. 记 $rH^T=(s_1,\ s_2)$。
2. 如果 $s_1=0$，那么没有错误（或者多于一个错误），算法结束。
3. 如果 $s_1\neq 0$，计算 s_2/s_1，将得到 α 的幂 α^{j-1}，错误在第 j 个位置。如果在有限域 $\mathbf{Z}_2$ 上考虑，那么算法结束，因为 $e_j=1$。但是对于其他的有限域，e_j 的取值还有一些其他的选择。
4. 计算 $e_j=s_1/\alpha^{(j-1)(k+1)}$。这是错误向量 e 的第 j 个元素，e 中其他元素都是 0。
5. 从收到的向量 r 中减去错误向量 e，就得到正确的码字 c。

举例：我们再考虑一下前面提到的有限域 $\mathbf{Z}_2$ 上设计距离为 7 且长度为 7 的 BCH 码。它是一个长度为 7 的二进制循环码，并且有两个码字：(0, 0, 0, 0, 0, 0, 0)，(1, 1, 1, 1, 1, 1, 1)。和前面一样设 α 是 X^3+X+1 的一个根，那么 α 是 1 的 7 次本原根。

在计算之前我们先要推导出一些关于 α 的幂的计算结论。我们有 $\alpha^3=\alpha+1$，将这个结论运用于 α 的幂中得到：

$$\alpha^4=\alpha^2+1,$$

$$\alpha^5=\alpha^3+\alpha^2=\alpha^2+\alpha+1,$$

$$\alpha^6=\alpha^3+\alpha^2+\alpha=(\alpha+1)+\alpha^2+\alpha=\alpha^2+1。$$

同样，结论 $\alpha^j=\alpha^{j \pmod 7}$ 也是有用的。

现在可以计算

$$rH^T=(1,1,1,1,0,1,1)\begin{pmatrix}1 & 1\\ \alpha & \alpha^2\\ \alpha^2 & \alpha^4\\ \vdots & \vdots\\ \alpha^6 & \alpha^{12}\end{pmatrix}$$

$$=(1+\alpha+\alpha^2+\alpha^3+\alpha^5+\alpha^6,1+\alpha^2+\alpha^4+\alpha^6+\alpha^{10}+\alpha^{12})$$

$$=(\alpha+\alpha^2,\alpha)。$$

例如，第一个元素的和可用如下方法估计出来：

$$1+\alpha+\alpha^2+\alpha^3+\alpha^5+\alpha^6=1+\alpha+\alpha^2+(1+\alpha)+(\alpha^2+\alpha+1)+(\alpha^2+1)=\alpha+\alpha^2。$$

因此 $s_1=\alpha+\alpha^2$，$s_2=\alpha$。我们需要计算 s_2/s_1，因为 $s_1=\alpha+\alpha^2=\alpha^4$，有

$$s_2/s_1=\alpha/\alpha^4=\alpha^{-3}=\alpha^4。$$

因此，$j-1=4$，故错误发生在 $j=5$ 的位置。错误向量的第 5 个元素是 $s_1/\alpha^4=1$，于是错误向量是（0，0，0，0，1，0，0）。纠错后的信息是

$$r-e=(1,1,1,1,1,1,1)。$$

■

下面是算法的原理。由于 $cH^T=0$，我们有

$$rH^T=cH^T=eH^T=eH^T=(s_1,s_2),$$

如果 $e=(0,0,...,e_j,0,...)$ 其中 $e_j\neq 0$，那么由 H 的定义得到

$$s_1=e_j\alpha^{(j-1)(k+1)},s_2=e_j\alpha^{(j-1)(k+2)},$$

因此，$s_2/s_1=\alpha^{j-1}$。同样可以得到 $s_1/\alpha^{(j-1)(k+1)}=e_j$。

16.9 Reed-Solomon 码

1960 年发明的 Reed-Solomon 码是 BCH 码的一个例子。由于它对某些特定类型的错误比较有效，故广泛应用在航天器的通信以及激光唱片中。

设 **F** 是一个有 q 个元素的有限域，并且 $n=q-1$。从有限域的定理我们有一个基本的结论，即 **F** 包含了一个 1 的 n 次本原根 α。在 $1\leq d<n$ 上选择 d，并设

$$g(X)=(X-\alpha)(X-\alpha^2)\cdots(X-\alpha^{d-1}),$$

这是一个以 **F** 中的元素为系数的多项式。它产生 **F** 域上的长度为 n 的 BCH 码 C，称之为 **Reed-Solomon 码**。

由于 $g(\alpha)=\cdots=g(\alpha^{d-1})=0$，BCH 边界指出码 C 的最小距离至少是 d。由于 $g(X)$ 是一个 $d-1$ 次的多项式，它有至多 d 个非零的系数，因此对应 $g(X)$ 系数的码字是一个权重最多为 d 的码字，进而得到 C 的最小权重是 d。C 的维数等于 $n-1-\deg(g)=n-1-d$，因此 Reed-Solomon 码是一个 $[n,n-1-d,d]$ 循环码。

C 中的码字对应的多项式是

$$g(X)f(X),其中 \deg(f)\leq n-d。$$

由于对每个 $f(X)$ 的 $n-d+1$ 个系数有 q 种选择，因此有 q^{n-d+1} 个这样的多项式 $f(X)$，并且 C 中有 q^{n-d+1} 个码字。故 Reed-Solomon 码是 MDS 码，也就是说，有 Singleton 边界条件（16.3 节）。

举例：设 $\mathbf{F}=\mathbf{Z}_7=\{0, 1, 2, ..., 6\}$ 是整数模 7 的余数[1]，那么 $q=7$ 并且 $n=q-1=6$。$\mathbf{F}$ 上的一个 1 的 6 次本原根 α 和前面的模 7 本原根相同（见 3.7 节）。于是可以取 $\alpha=3$，选择 $d=4$，那么

$$g(X)=(X-3)(X-3^2)(X-3^3)=X^3+3X^2+X+6。$$

这个码有生成矩阵

$$G=\begin{pmatrix}6 & 1 & 3 & 1 & 0 & 0\\ 0 & 6 & 1 & 3 & 1 & 0\\ 0 & 0 & 6 & 1 & 3 & 1\end{pmatrix},$$

在这个码中有 $7^3=343$ 个码字，可以通过对 G 中的 3 行进行模 7 线性变换得到，码的最小权重是 4。 ■

举例：设 $\mathbf{F}=GF(4)=\{0, 1, \omega, \omega^2\}$，同 3.10 节中所描述的一样。那么 $\mathbf{F}$ 有 4 个元素，$n=q-1=3$，并且 $\alpha=\omega$。选择 $d=2$，于是有

$$g(X)=(X-\omega),$$

矩阵

$$G=\begin{pmatrix}\omega & 1 & 0\\ 0 & \omega & 1\end{pmatrix}$$

是码的生成矩阵。这个码包含所有 G 中两行的 16 个线性变换。例如

$$\omega\cdot(\omega,1,0)+1\cdot(0,\omega,1)=(\omega^2,0,1),$$

码字的最小权重是 2。 ■

在许多应用中，错误并不是随机分布的，相反是突发的。例如，在一张 CD 中，刮痕将造成许多邻近比特的错误。一次太阳能量的突发将对宇宙飞船的通信造成同样的影响。Reed-Solomon 码对这类情况很有效。

例如，假设 $\mathbf{F}=GF(2^8)$。$\mathbf{F}$ 中的元素由字节组成，每个字节 8 个比特，如 3.10 节所描述的那样。我们有 $n=2^8-1=255$，设 $d=33$，那么码字是包含 255 字节的向量，其中有 222 个消息字节和 33 个纠错字节。这些码字被作为 $8\times255=2040$ 的比特流传送，传输过程中的干扰将破坏其中的一些比特。然而，由于是突发的干扰，这些错误比特将限制在传输比特流中很小的范围内。比如，假设干扰位都在 121（$=15\times8+1$）个连续的干扰位中，那么最多可能有 16 字节的错误。因此，这些错误可以被检测到（因为 $16<d/2$）。另一方面，如果 121 比特随机地分布在 2040 个比特流中，大量的字节可能发生错误，纠错将是不可能的。因此，码的选择取决于预期可能出现的错误类型。

16.10 McEliece 密码体制

在本书中，我们主要基于数论的原理描述密码体制，还有另外的基于其他复杂问题的密

1 或称模 7 整数——译者注

码体制。这里我们将介绍一种基于一个二进制线性码寻找最近码字的困难性的密码体制。

思想很简单。假设有一个长度为 1024 位的二进制串，其中有 50 个错误。对于这些错误共有 $\binom{1024}{50} \approx 3 \times 10^{85}$ 种可能的位置，因此对于所有的可能性进行穷尽搜索是不可能的。然而，假设你有一个不为人知的高效算法，那么只有你可以纠正这些错误并找到正确的串。McEliece 显示了怎样用这个原理得到一个公钥密码体制。

鲍勃选择 G 作为一个 (n, k) 线性纠错码 C 的生成矩阵，其中 $d(C) = d$。他选择 S 是一个 $k \times k$ 的模 2 可逆矩阵，选择 P 是一个 $n \times n$ 的置换矩阵。这意味着 P 的每一行和每一列都只有一个 1，其他所有元素都是 0。定义

$$G_1 = SGP,$$

矩阵 G_1 是密码体制的公钥。鲍勃秘密地保存 S, G, P。

艾丽斯为了给鲍勃发送一个消息 x，她产生一个长度为 n 权重为 t 的随机二进制串，并通过计算

$$y \equiv xG_1 + e \pmod 2$$

构造密文。鲍勃按照下面的步骤对 y 解密：

1. 计算 $y_1 \equiv yP^{-1}$。（由于 P 是一个置换矩阵，$e_1 = eP^{-1}$ 仍然是一个权重为 t 的二进制串，我们有 $y_1 \equiv xSG + e_1$。）
2. 对 y_1 运用码 C 的纠错解码器以纠正“错误”，并得到与 y_1 最近的码字 x_1。
3. 计算 x_0 使 $x_0G \equiv x_1$ 成立（在所考虑的这个例子中，x_0 只是简单地取 x_1 的前 k 个比特）。
4. 计算 $x \equiv x_0S^{-1}$。

体制的安全性取决于对 y_1 解码得到 x_1 的困难程度。S 给体制增加了一点安全性，但是一旦知道了由 GP 产生的码的解码算法，选择明文攻击就能够得到矩阵 S（正如希尔密码一样）。

为了使解码困难，应该选择较大的 $d(C)$。McEliece 建议采用一个 [1024, 512, 101] Goppa 码。**Goppa 码**有 $n = 2^m$, $d = 2t + 1$, $k = n - mt$ 形式的参数。例如取 $m = 10$, $t = 50$，产生刚才提到的 [1024, 512, 101] 码，它能够纠正最多 50 个错误。对于给定的 m, t，实际上有许多带这样参数的不同 Goppa 码。这里我们只讨论它们中有高效解码能力的算法，因而它们能用来快速地纠错。

举例：矩阵

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix},$$

是 [7, 4] 汉明码的生成矩阵。假设艾丽斯想传送消息

$$m = (1,0,1,1)$$

给鲍勃。为了达到这个目的，鲍勃需要生成一个可逆的矩阵 S 和一个随机的置换矩阵 P，并且秘密地保存。如果鲍勃选择

$$S = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

以及

$$P=\begin{pmatrix}0&0&1&0&0&0&0\\1&0&0&0&0&0&0\\0&0&0&0&1&0&0\\0&0&0&0&0&1&0\\0&0&0&0&0&0&1\\0&1&0&0&0&0&0\\0&0&0&1&0&0&0\end{pmatrix},$$

通过这些鲍勃能够得到公共的加密矩阵：

$$G_1=\begin{pmatrix}0&0&1&1&0&1&0\\1&0&1&0&0&1&1\\1&1&0&0&0&1&0\\1&0&1&0&1&0&0\end{pmatrix}。$$

为了解密，艾丽斯产生她自己的随机错误向量 e，计算密文 $y=xG_1+e$。在这个汉明码的例子中，错误向量的权重是 1。假设艾丽斯选择

$$e=(0,1,0,0,0,0,0),$$

那么

$$y=(0,0,0,1,1,0,0)。$$

鲍勃解密时首先计算

$$y_1=yP^{-1}=(0,0,1,0,0,0,1),$$

通过运用奇偶校验矩阵计算伴随式并且改变对应的比特位得到

$$x_1=(0,0,1,0,0,1,1)。$$

然后鲍勃构造一个满足 $x_0G=x_1$ 的 x_0，这可以通过提取 x_1 中的前 4 个元素得到，即

$$x_0=(0,0,1,0)。$$

鲍勃通过计算

$$x=x_0S^{-1}=(1,0,1,1)$$

解密，得到最初的明文。 ■

McEliece 密码体制看上去非常安全。对于其安全性的讨论参见 [Chabaud]。这个体制相对于 RSA 来说有一个缺点是它的公钥 G_1 太大。

16.11 其他问题

纠错码是数学领域和工程领域都在研究的一个重大课题。在这一章里我们仅仅讨论了该领域中少数的一些概念。纠错码的许多相关领域都没有讨论。

也许纠错码中最重要的是卷积码。本章中我们仅仅讨论了分组码，尤其是数据被分为固定长度为 k 并映射为长度为 n 的码字。然而，在许多应用中，数据是以连续的形式产生的，并且很容易将数据流映射为编码符号流。比如，这类码在编码或解码前不需要为等待完整的符号块而进行延时。

另一个关于纠错码的主题是解码效率。对于线性码我们采用伴随式解码，这要比寻找最近的码字效率高。然而，对于大的线性码，伴随式解码的效率也很低，不能用于实际中。当 BCH 码和 Reed-Solomon 码发明后，前面的解码算法对于有多个错误的情况不再实用。后来 Berlekamp 和 Massey 提出了解 BCH 码和 Reed-Solomon 码的有效方法。关于这个课题还有许多研究，我们介绍给读者一些书以便以后深入研究解码，[Lin-Costello]，[Wicker]，[Gallager] 和 [Berlekamp]。

我们仅仅考虑了比特或符号错误。然而，在现在的计算机网络中，错误的类型不仅仅是比特和符号错误，还有数据段的完全丢失。比如在互联网上，数据以包的形式在网上传输。由于网络上各个位置的拥塞，比如路由器和交换机，数据包可能丢失并且永远不能到达预定的接收者。在这种情况下，接收者要通知发送者重发数据包。传输控制协议（TCP）提供了传输丢失包的机制。

在构造加密体制时，结合各种差错控制机制以保证加密信息的可靠传输是至关重要的，否则接收者可能无法解密收到的信息。

最后，我们提到编码理论和各种数学问题有着重要的联系，比如寻找高次球的密集填充，更多内容见 [Thompson]。

16.12 习　　题

1. 两个 [7, 4] 汉明码的码字被传输，收到的是 0100111 和 0101010。每个码字最多包含一个错误。纠正这些错误，并且确定与矩阵 G 相乘得到码字的 4 比特信息。

2. 一个 ISBN 码被错误地写成了0-13-116093-8，说明这不是一个正确的 ISBN 码。找到两个有效的不同的 ISBN 码，使其满足发生一位错误能给出这个数字。这说明了 ISBN 码不能纠错。

3. 下面是一个二进制 $[n, k]$ 码 C 的奇偶校验矩阵：

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{pmatrix}。$$

(a) 找到 n 和 k。

(b) 找到 C 的生成矩阵。

(c) 列出 C 中的码字。

(d) C 的编码率是多少？

4. 设 $C = \{(0, 0, 0)\ (1, 1, 1)\}$ 是一个二进制循环码。

(a) 找到 C 的奇偶校验矩阵。

(b) 列出 C 的陪集和陪集首。

(c) 找出每个陪集的伴随式。

(d) 使用伴随式解码方法解码消息 (1, 1, 0)。

5. 设 C 是二进制码 $\{(0, 0, 0)\ (1, 1, 0)\ (1, 0, 1)\ (0, 1, 1)\}$。

(a) 证明 C 是非线性的。

（b）$d(C)$ 是多少？（由于 C 是非线性的，所以 $d(C)$ 不能通过计算最小权重得到。）

（c）证明 C 以等号满足 Singleton 边界条件。

6. 证明权重函数（$\mathbf{F}^n$上的）满足三角不等式：$wt(u+v) \leqslant wt(u)+wt(v)$。

7. 证明 $A_q(n,n)=q$，其中 $A_q(n,d)$ 是 16.3 节中定义的函数。

8. 设 C 是长度为 n 的循环码，证明 $C^\perp$是长度为 n 的奇偶校验码。（对于任意的 $\mathbf{F}$ 成立。）

9. 设 C 是一个线性码，并且 $u+C$ 和 $v+C$ 是 C 的陪集。证明当且仅当 $u-v\in C$ 时 $u+C=v+C$。（提示：为了证明 $u+C=v+C$，需要证明对于每个 $c\in C$ 有 $u+c\in v+C$，为了证明相反的蕴涵式，需要用到 $u\in u+C$。）

10. 证明：如果 C 是一个自对偶的 $[n,\ k,\ d]$ 码，那么 n 必定是偶数。

11. 证明：$g(X)=1+X+X^2+\cdots+X^{n-1}$ 是 $[n,\ 1]$ 循环码的生成多项式。（对于任意的 $\mathbf{F}$ 都成立。）

12. 设 $g(X)=1+X+X^3$是以 $\mathbf{Z}_2$中元素为系数的多项式。

（a）证明 $g(X)$ 是 X^7-1 在 $\mathbf{Z}_2[X]$ 中的一个因子。

（b）多项式 $g(X)$ 是一个循环 $[7,\ 4]$ 码 C 的生成多项式，找到 C 的生成多项式。

（c）找到 C 的一个奇偶校验矩阵 H。

（d）证明 $G'H^T=0$，其中

$$G'=\begin{pmatrix}1&1&0&1&0&0&0\\0&1&1&0&1&0&0\\0&0&1&1&0&1&0\\0&1&1&1&0&0&1\end{pmatrix}。$$

（e）证明 G'的行产生码 C。

（f）证明 G'中列的置换给出了 $[7,\ 4]$ 汉明码的生成多项式，因此这两个码是相等的。

13. 设码 C 是长度为 4 的循环码，生成多项式 $g(X)=X^2+1$，下面哪个多项式对应 C 中的元素？

$$f_1(X)=1+X+X^3,\ f_2(X)=1+X+X^2+X^3,\ f_3(X)=X^2+X^3。$$

14. 设 $g(X)$ 是长度为 n 的循环码 C 的生成多项式，并且设 $g(X)h(X)=X^n-1$。记 $h(X)=b_0+b_1X+\cdots+X^\ell$。证明对偶码 $C^\perp$ 是循环码，并且生成多项式 $\tilde{h}_r(X)=(1/b_0)(1+b_{\ell-1}X+\cdots+b_1X^{\ell-1}+b_0X^\ell)$（因子 $1/b_0$用来使最高的非零系数是 1）。

15.（a）设 C 是一个长度为奇数 n 的二进制循环码（也就是说，C 包含两个向量，一个是全 0，一个是全 1），证明 C 是一个理想码。（提示：证明每个向量都在一个球区域内，这个球是半径为 $(n-1)/2$ 的两个球区域之一。）

（b）用（a）证明如果 n 是奇数，那么 $\sum_{j=0}^{(n-1)/2}\binom{n}{j}=2^{n-1}$。（这也能够通过将二项式理论应用于 $(1+1)^n$，并且注意到我们使用了一半的项目来证明。）

16. 设 $2\leqslant d\leqslant n$ 并且设 $V_q(n,\ d-1)$ 描述了半径为 $d-1$ 的汉明球中点的数量。Gilbert-Varshamov 边界条件的证明是构造一个 $(n,\ M,\ d)$ 码，其中 $M\geqslant q^n/V_q(n,d-1)$，但是这个码很可能是非线性的。本练习将构造一个线性 $[n,\ k,\ d]$ 码，其中 k 是满足 $q^k\geqslant q^{n-1}/V_q(n,d-1)$ 的最小整数。

（a）证明存在一个 $[n,\ 1,\ d]$ 码 C_1。

(b) 假设 $q^{j-1} < q^n / V_q(n, d-1)$，并且已经构造了一个域 $\mathbf{F}^n$ 上的 $[n, j-1, d]$ 码 C_{j-1}（其中 $\mathbf{F}$ 是有 q 个元素的有限域）。证明对于所有的 $c \in C_{j-1}$，存在一个向量 v 使得 $d(v,c) \geq d$。

(c) 设 C_j 是 v 和 C_{j-1} 生成的子空间。证明 C_j 是 j 维的，并且 C_j 中的每个元素都可以写成 $av + c$ 的形式，其中 $a \in \mathbf{F}$，$c \in C$。

(d) 设 $av + c$ 是 C_j 的一个非零元素，证明 $wt(av + c) = wt(v + a^{-1}c) = d(v, -a^{-1}v) \geq d$。

(e) 证明 C_j 是一个 $[n, j, d]$ 码。继续通过归纳得到预期的码 C_k。

(f) 这里有一个知识点，实际上我们构造了一个 $[n, k, e]$ 码，其中 $e \geq d$。证明通过改变 (b) 中的 v，可以满足对于某些 $c \in C_{j-1}$ 有 $d(v, c) = d$，因此我们得到一个 $[n, k, d]$ 码。

17. 证明 Golay 码 $\mathcal{G}_{23}$ 是理想的。

18. 设 α 是多项式 $X^3 + X + 1 \in \mathbf{Z}_2[X]$ 的一个根。

(a) 利用 $X^3 + X + 1$ 能除尽 $X^7 - 1$，证明 $\alpha^7 = 1$。

(b) 证明 $\alpha \neq 1$。

(c) 假设 $\alpha^j = 1$，其中 $1 \leq j < 7$，那么 $\gcd(j,7) = 1$，因此存在整数 a，b 使 $ja + 7b = 1$。用这些证明 $\alpha^1 = 1$ 是一个矛盾式，从而证明 α 是一个 1 的 7 次本原根。

19. 设 C 是由多项式 $g(X) = 1 + X^2 + X^3 + X^4$ 产生的二进制码。同 16.8 节中一样，$g(1) = g(\alpha) = 0$，其中 α 是 $X^3 + X + 1$ 的一个根。假设收到了消息 (1, 0, 1, 1, 0, 1, 1)，其中有一个错误，用 16.8 节中的结论纠正这个错误。

20. 设 $C \subset \mathbf{F}^n$ 是长度为 n 的循环码，生成多项式为 $g(X)$。假设 $C \neq \mathbf{F}^n$。

(a) 证明 $\deg(g) \geq 1$。

(b) 记 $X^n - 1 = g(X)h(X)$，设 α 是一个 1 的 n 次本原根，证明 $1, \alpha, \alpha^2, \ldots, \alpha^{n-1}$ 中至少有一个是 $h(X)$ 的根。(可以使用结论：$h(X)$ 不可能有多于 $\deg(h)$ 个根。)

(c) 证明 $d(C) \geq 2$。

16.13 上机题

1. 来自 Golay 码 $\mathcal{G}_{24}$ 的 3 个码字被传送，并且收到向量

$$(0,1,0,0,0,0,1,1,0,1,0,1,1,0,1,1,0,1,0,0,0,0,1,1),$$
$$(0,0,1,0,1,0,1,0,1,0,1,0,0,1,0,0,1,1,1,0,1,1,0,0),$$
$$(1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,1)。$$

纠正其中的错误。(Golay 矩阵作为 *golay* 保存，矩阵 B 作为 *golayb* 保存。)

2. 一个 11 比特的消息乘以汉明 [15, 11] 码的生成矩阵，并且得到的码字被发送。接收者接收到向量

$$(0,1,1,0,0,0,1,1,0,0,0,1,0,1,0),$$

假设最多存在一个错误，请纠正它并确定最初 11 比特的信息。(汉明 [15, 11] 码的奇偶校验矩阵作为 *hammingpc* 存储。)

第17章 密码学中的量子技术

量子计算是最近才开始兴起的一个新的研究领域。量子计算和量子密码学是在怎样将量子力学原理运用在完成计算的研究中产生的。1982年诺贝尔奖获得者理查德·弗曼观察到一些量子力学现象不能在传统的计算机上进行仿真。他认为那些在传统计算机上不能进行的计算可以运用量子力学来解决。弗曼没有提出任何关于这种装置甚至更小版本的例子，直到最近才在构造上有了一点进步。

1994年量子计算领域有了重大的突破，原因在于美国电话电报公司研究工作实验室提出了一种可以因数分解（大概的）多项式中的整数的算法（如果一台相应的量子计算机被创造出来）。用这种算法实现的第一个例子是一个巨大的成就，在这个例子中量子技术远远地超过了传统的计算技术。

本章将介绍来自量子计算和量子密码领域的两个例子。毫无疑问本章是对这个新领域的彻底论述。当我们写这一章时，在NIST和其他方面又有了重大的突破，显然这个领域将持续快速地发展。

关于量子计算的许多书和说明文章正在被撰写，其中一本比较容易读的参考书是[Rieffel-Polak]。

17.1 一个量子实验

自从量子力学用来处理我们日常经验不适用的一些概念后，它就成了对于非物理学者来说很难解释的一门学科。特别地，量子力学现象发生的范围是在只有通过专门的设备才能观察到的原子层。然而，也有一些容易被我们理解的例子。现在提出了一个这样的例子，通过它来发展需要描述某些量子计算协议的数学公式。

既然量子力学是基于微粒层的物理学，那么就需要可被观察到的微粒。光子是一种可以组成光的微粒，因此可以被观察到（用其他微粒——如电子的类似实例也可以实现，但需要更加复杂的设备）。

为了更好地理解这个实验，我们推荐在家里试验一下。首先准备一个强光源（如一个强光手电筒或一个激光器）和3个从照相器材店里买来的人造偏光板滤光器。

3个滤光器分别标记为A、B、C。旋转它们，使之出现下列偏振现象：水平、45°、互相垂直（实验后将详细解释偏振现象）。将光照向墙壁，如图17.1所示在光源与墙壁之间插入滤光器A。光子穿过滤光器将有水平偏振现象。如图17.2所示，再插入滤光器C。由于

滤光器 C 有垂直偏振现象，它过滤出所有来自滤光器 A 的水平偏振光子。注意，执行完以上几步后没有光到达墙壁，这两个滤光器过滤掉了光的所有成分。现在进行最后一步（最奇异的一步），在滤光器 A 和 C 之间插入滤光器 B。如图 17.3 所示，应该可以观察到有光到达墙壁。这很让人疑惑，因为滤光器 A 和 C 足以过滤掉所有的光，而多余的第三个滤光器反而允许光到达墙壁。

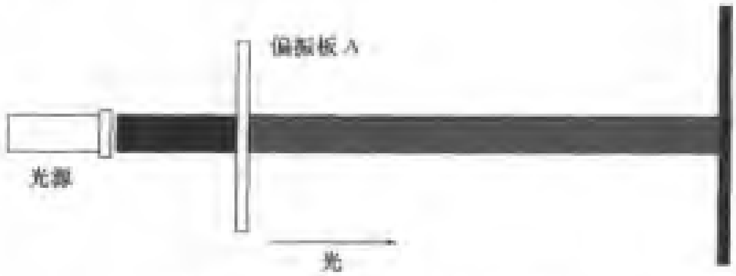


图 17.1　仅仅插入滤光器 A 的光子实验

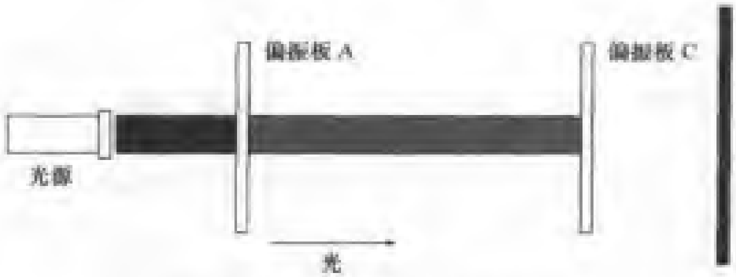


图 17.2　插入滤光器 A 和 C 的光子实验

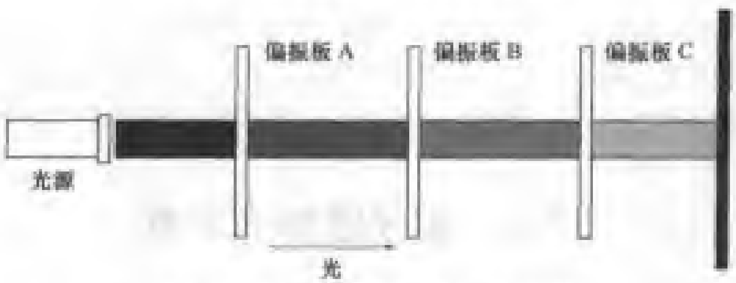


图 17.3　所有滤光器被插入后的光子实验

为了解释这个实例，我们有必要讨论一下光的偏振现象的概念。

光是一种电磁波，这意味着它是由正交传播到磁场的电场组成的。为了使之更形象，我们假设光沿着 x 轴传播。例如，假设电场是位于 xz 平面的一种波状物质，那么相应地，磁场是 xy 平面的波状物质。对于这样的场景，光被称作垂直偏振。大体上，偏振沿着电场所在的方向。在这个方向上没有约束。

我们将通过二维复向量空间中的一个单位向量来描述一个光子的偏振（然而，针对我们所讨论的目的，在实数范围内就足够了）。这个向量空间有一个点积，由公式：$(a, b)\cdot(c, d)=a\bar{c}+b\bar{d}$给出，其中 $\bar{c}$ 和 $\bar{d}$ 表示 c 和 d 的共轭复数，那么向量 (a, b) 的长度的平

方为 $(a, b) \cdot (a, b) = |a|^2 + |b|^2$。我们用 $|\uparrow\rangle$ 和 $|\rightarrow\rangle$ 来作为这个向量空间的一个基。用来自物理学的态矢（括号的第二部分）符号来表示向量。我们可以认为 $|\uparrow\rangle$ 表示垂直方向，而 $|\rightarrow\rangle$ 表 示水平方向。因此，一个任意的偏振可以表示成 $a|\uparrow\rangle + b|\rightarrow\rangle$，其中 a 和 b 是复数。由于正在处理的是单位向量，其包含下面的属性：$|a|^2 + |b|^2 = 1$。当然我们也可以选择不同的正交基，如相应的45°旋转：$|\nwarrow\rangle$ 和 $|\nearrow\rangle$。

人造偏光板对光子的极性进行了测量后，有两种可能的结果：或者光子与滤光器相一致，或者光于与滤光器的方向垂直。如果向量 $a|\uparrow\rangle + b|\rightarrow\rangle$ 被一个垂直滤光器测量，那么光子通过滤光器后有垂直极性的概率为 $|a|^2$，有水平极性的概率为 $|b|^2$。

类似地，假设我们测量一个与45°的滤光器垂直的光子，由于

$$|\uparrow\rangle = \frac{1}{\sqrt{2}}|\nwarrow\rangle + \frac{1}{\sqrt{2}}|\nearrow\rangle,$$

所以光子通过滤光器（这意味着它被定位在45°的位置进行测量）的概率为 $(1/\sqrt{2})^2 = 1/2$。同样的方法可知，它不能通过滤光器（这意味着它在 -45°被测量）的概率也是1/2。

量子力学的基本原理之一就是通过这样一种测量法强迫光子进入到一定的状态。测量后，光子的状态将被改变为测量的结果。因此，如果测量 $a|\uparrow\rangle + b|\rightarrow\rangle$ 的状态为 $|\rightarrow\rangle$，那么从这一刻起，光子将处于状态 $|\rightarrow\rangle$。如果继续用一个 $|\rightarrow\rangle$ 滤光器来测量这个光子，将总是看到它处于 $|\rightarrow\rangle$ 状态；但是，如果用一个 $|\uparrow\rangle$ 滤光器来测量，将永远不会观察到这个光子处于 $|\uparrow\rangle$ 状态。

现在让我们解释一下这个实验。原始光在随机的偏振中被发射，这意味着光子以状态 $a_1|\uparrow\rangle + b_1|\rightarrow\rangle$ 被发射的概率等同于以状态 $a_2|\uparrow\rangle + b_2|\rightarrow\rangle$ 被发射的概率。被发射的光子仅仅有半数将通过 $|\rightarrow\rangle$ 滤光器，而且所有这些光子的状态将转变为 $|\rightarrow\rangle$（其余的一半被吸收或者被反射并转换为 $|\uparrow\rangle$）。当我们把垂直滤光器放在水平滤光器的后面时，处于状态 $|\rightarrow\rangle$ 的光子将被阻止。

当在中间插入滤光器 B 时，它相应地去测量 $|\nearrow\rangle$，那些有 $|\rightarrow\rangle$ 极性的光于以50%的概率变成 $|\nearrow\rangle$ 极性。因此，到达并通过 B 滤光器的光子数量会有25%的减少。现在 $|\nearrow\rangle$ 极性的光子也以50%的概率通过 $|\uparrow\rangle$ 滤光器，所以到达墙壁的光的总强度是原始强度的1/8。

17.2 量子密钥的分发

既然我们已经对量子力学有了一些了解，那么可以用它们来描述一种通过量子信道分配比特位的方法。这些比特位可以用来建立通过一个常规信道通信的密钥或其他的共享密钥。

我们首先描述一个量子位。先建立一个二维复向量空间，选择两个长度为1的正交向量；分别称为 $|0\rangle$ 和 $|1\rangle$。例如，这两个向量可以是前一节用到的两对正交向量中的任意一对。**量子位（quantum bit）**也可简称为**qubit**，是这个向量空间的一个单位向量。基于目前我们讨论的目的，可以把一个量子位看作是一个偏振的光子。我们已经选择符号 $|0\rangle$ 和 $|1\rangle$ 来方便地分别表示0和1位。另外的量子位则是这两位的线性结合。

既然量子位是单位向量，那么它能够表示成 $a|0\rangle + b|1\rangle$，其中 a 和 b 是满足条件 $|a|^2 + |b|^2 = 1$ 的复数。正如前一节中的光子一样，可以以基 $|0\rangle$ 和 $|1\rangle$ 来测量这个量子位。我们观

察到它在$|0\rangle$状态的概率为$|a|^2$。

现在来研究一下艾丽斯和鲍勃之间为了发送消息是怎样互相沟通的。他们需要两个条件：量子信道和常规信道。通过量子信道可以交换那些与环境交互相隔离的偏振光子（也就是说，环境不能改变光子）。常规信道用来互相传送普通的信息。假设别有用心的观察者伊芙能够观察到常规信道中被传送的东西，而且她能够在量子信道中观察并再次发送光子。

艾丽斯通过向鲍勃传送一串比特位开始建立信息。如下所示用一个每一位都是随机选择的基对它们进行编码。有两个数：$B_1=\{|\uparrow\rangle,|\rightarrow\rangle\}$ 和 $B_2=\{|\nwarrow\rangle,|\nearrow\rangle\}$。如果艾丽斯选择$B_1$，那么她把 0 编码为$|\uparrow\rangle$，把 1 编码为$|\rightarrow\rangle$，同样地，如果她选择$B_2$，那么她用$B_2$中的两个元素来编码 0 和 1。

每次艾丽斯传送一个光子，鲍勃随机地选择基数B_1或者B_2来测量。因此，对每一个光子，他得到一个所选择的基数的元素作为测量结果。鲍勃记录他得到的测量结果并保守秘密，然后他告诉艾丽斯他测量每个光子时所用的基数。作为回应，艾丽斯告诉他对于她所传送的光子哪一个基数是正确的。他们保留那些用相同基数的位，而丢掉其他的位。由于用了两种基数，所以艾丽斯和鲍勃将对艾丽斯所传送的位的大约半数达成一致。然后他们就可以用这些位作为一个常规密码体制的密钥。

例如：假设艾丽斯想传送比特位：0，1，1，1，0，0，1，0。她随机地选择基：B_1，B_2，B_1，B_1，B_2，B_2，B_1，B_2。因此，她传送量子位（光子）

$$|\uparrow\rangle,|\nearrow\rangle,|\rightarrow\rangle,|\rightarrow\rangle,|\nwarrow\rangle,|\nwarrow\rangle,|\rightarrow\rangle,|\nwarrow\rangle$$

给鲍勃。他选择基：B_2，B_2，B_2，B_1，B_2，B_1，B_1，B_2，测量艾丽斯传送的量子位，并且告诉艾丽斯他用的基。艾丽斯告诉他第 2，4，5，7，8 个与她选择的相匹配。这些使鲍勃得到了测量结果：

$$|\nearrow\rangle,|\rightarrow\rangle,|\nwarrow\rangle,|\rightarrow\rangle,|\nwarrow\rangle,$$

并且它们与比特位 1，1，0，1，0 相对应。因此艾丽斯和鲍勃都有相同的字符串 1，1，0，1，0，他们把 11010 作为以后通信的密钥（例如：如果他们得到了一个较长的字符串，那么他们可以用前 56 位字符作为一个 DES 密钥）。■

量子密钥分配的安全性基于量子力学的规律和下面的这个微粒的基本原理：微粒的状态会被改变。既然偷听者伊芙为了观察到艾丽斯和鲍勃之间传送的光子必须进行测量，那么伊芙将在艾丽斯和鲍勃达成一致的数据中引进错误。

让我们看看这是怎样发生的。假设伊芙测量由艾丽斯传送的光子的状态，并允许这些被测量的光子继续到达鲍勃。既然这些光子被伊芙测量了，那么它们将变成伊芙观察到的状态。当伊芙进行测量时，她有一半的可能将用一个错误的基数。当鲍勃进行测量时，如果他用正确的基数，那么有 25% 的可能他将测出错误的值。

我们来更加详细地研究一下上面的论述。假设艾丽斯传送了一个与$|\rightarrow\rangle$相应的光子，而且鲍勃用了与艾丽斯相同的基数B_1。如果伊芙用B_1，那么光子正确地通过了并且鲍勃能够正确地测量光子。然而，如果伊芙用B_2，那么她同样地能够测量$|\nearrow\rangle$和$|\nwarrow\rangle$。传到鲍勃的光子拥有这些方向的其中之一，因此他有一半的可能能够正确地测量为$|\rightarrow\rangle$，还有一半的可能不能正确地测量。结合这两种可能的基数选择，伊芙导致鲍勃有 25% 的可能测量出错误的值。

因此，任何偷听都在艾丽斯和鲍勃之间的通信中引入了较高的错误率。如果艾丽斯和鲍

勒通过常规的信道测试他们数据的差异，那么将探测到任何的偷听。

这种技术实际已经应用到了在超过 24 千米的距离利用常规的光纤纤维电缆上创建密钥 [Hughes et al.]。

17.3 Shor 算法

量子计算机还没实现。目前的版本仅仅能处理一些量子位。但是，如果能够解决这个技术难题并且大型的量子计算机被建立起来，那么将对密码学产生十分巨大的影响。本节我们将通过 Peter Shor 提出的算法对量子计算机怎样因数分解大整数给出一个简捷的描述。我们尽量避免讨论量子力学，并请读者相信一个量子计算机能够迅速地执行我们描述的所有操作。至于更多的细节，可以参考其他资料，例如 [Ekert-Josza] 和 [Rieffel-Polak]。

什么是量子计算机？它能做什么？首先，我们来看看传统的计算机能做什么。例如，它接受一个二进制的输入——100010，并给出一个二进制的输出——可能为 0101。如果它有几个输入，那么必须分别地处理它们。量子计算机把某一数目的量子位作为输入，并输出一些量子位。主要的不同在于输入和输出的量子位可以是某些基本状态的线性组合。量子计算机同时在这个线性组合里对所有的基本状态进行操作。实际上，量子计算机是一个大型的并行机。

例如：用基本状态 $|100\rangle$ 来表示 3 个微粒，第一个在方向 1，后两个在方向 0（指贯穿全文隐含地指定的一些基数）。量子计算机能够接受 $|100\rangle$ 并产生一些输出。然而，当在基本状态工作时，它也能够把诸如

$$\frac{1}{\sqrt{3}}(|100\rangle + |011\rangle + |110\rangle)$$

这样的基本量子状态的规格化线性组合当作输入，并且尽可能快地产生一个输出。毕竟，没有进行测量，计算机不能分辨一个量子状态是基本状态之一还是它们的线性组合，但是这样的测量将改变输入。正是这种与状态的线性组合同时工作的能力潜在地使量子计算机非常强大。

假设有一个能够通过常规计算机对输入值 x 求值的函数 $f(x)$。常规计算机要求一个输入产生一个输出。恰恰相反，量子计算机能够把所有可能输入状态的总和

$$\frac{1}{C}\sum_{x}|x\rangle$$

（C 为一个规格化因子）作为输入，并产生输出

$$\frac{1}{C}\sum_{x}|x,f(x)\rangle,$$

其中 $|x,f(x)\rangle$ 是一个量子位的长序列，表示 x 和 $f(x)$ 的值（技术点：为了使一些微粒变成 $f(x)$，最好输入 $(1/C)\sum_{x}|x,00\cdots\rangle$，但为了简单起见，我们将不这样做），所以我们能获得所有 $f(x)$ 值的列表。这看起来非常好，但是有一个问题，就是如果进行一个测量，那么将强迫量子状态变成测量的结果。对于随机选择的 x_0，得到 $|x_0, f(x_0)\rangle$，而结果中的其他状态被毁掉了。所以，如果要看 $f(x)$ 值的列表，最好认真地做它，因为只有一

次机会。特别地，为了把输出转化为一种更加合适的形式，可能要对它进行某些转变。设计量子计算机的技巧在于，设计计算使要检查的输出看起来比其他的具有更高的可能性。这就是所有在 Shor 的因数分解法则中所包含的内容。

17.3.1　因数分解

我们要对 n 进行因数分解。策略如下所示，如果能找到（非平凡的）a 和 r 满足 $a^r \equiv 1 \pmod n$，那么就记住它们，然后会有很大的机会对 n 进行因数分解（看一下 6.4 节的指数因数分解方法）。选择一个随机数 a 并考虑序列 $1, a, a^2, a^3, \ldots \pmod n$。如果 $a^r \equiv 1 \pmod n$，那么由于 $a^{j+r} \equiv a^j a^r \equiv a^j \pmod n$，所以这个序列的元素每间隔 r 就重复出现。如果能够测量出这个序列的周期（或者周期的倍数），就能获取满足条件 $a^r \equiv 1 \pmod n$ 的数 r。因此我们要把量子计算机设计为可满足对输出进行测量时，就有较大的可能性获取周期的情况。

17.3.2　离散的傅立叶变换

我们需要一种方法来获取周期性序列的周期。以前傅立叶变换可以达到这个目的，而且在目前的形式下也可以使用。假设有一个长度为 2^m 的序列

$$a_0, a_1, \ldots, a_{2^m-1},$$

其中 m 为某个整数。定义傅立叶变换为

$$F(x) = \frac{1}{\sqrt{2^m}} \sum_{c=0}^{2^m-1} e^{\frac{2\pi i c x}{2^m}} a_c, \text{其中 } 0 \leq x < 2^m。$$

例如，考虑长度为 8、周期为 4 的序列

$$1, 3, 7, 2, 1, 3, 7, 2$$

长度除以周期等于频率，也就是 2，频率是指序列重复的次数。通过傅立叶变换可以得到值

$$F(0) = 26/\sqrt{8}, \quad F(2) = (-12 + 2i)/\sqrt{8},$$

$$F(4) = 6/\sqrt{8}, \quad F(6) = (-12 - 2i)/\sqrt{8},$$

$$F(1) = F(3) = F(5) = F(7) = 0。$$

例如，让 $\zeta = e^{2\pi i/8}$，我们发现

$$\sqrt{8}F(1) = 1 + 3\zeta + 7\zeta^2 + 2\zeta^3 + \zeta^4 + 3\zeta^5 + 7\zeta^6 + 2\zeta^7。$$

因为 $\zeta^4 = -1$，消除相应项，可以得到 $F(1) = 0$。F 的非零值出现在 2 的倍数处，也就是频率的倍数处。

让我们考虑另一个例子：2, 1, 2, 1, 2, 1, 2, 1。傅立叶变换后，

$$F(0) = 12/\sqrt{8}, \quad F(4) = 4/\sqrt{8},$$

$$F(1) = F(2) = F(3) = F(5) = F(6) = F(7) = 0。$$

这里 F 的非零值又出现在频率的倍数处。

一般来说，如果周期是 2^m 的约数，那么 F 的所有非零值将出现在频率的倍数处（虽然这样，频率的倍数仍然能够趋向于 0）。这可以参考练习 2。

假设周期不是2^m的约数，我们来看下面的例子。对于序列1，0，0，1，0，0，1，0，它的长度为8，周期和频率近似为3。在图17.4中，我们标明了它的傅立叶变换（由于这些是实数，因此比傅立叶变换的复数值更容易画出）的绝对值。注意到在0，3，5处有峰值，如果继续对比较大的x值计算$F(x)$，那么将在8，11，13，16，... 处得到峰值。这些峰值间隔的平均值为8/3。序列的长度除以平均长度得到周期为8/（8/3）=3，这与我们的直觉是一致的。

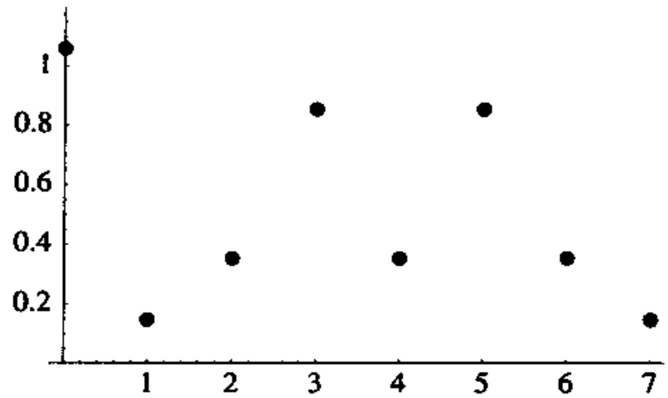


图17.4 离散傅立叶变换的绝对值

实际上，在0点处有一个峰值没有什么奇怪的。傅立叶变换法则说明0点的值仅仅是序列长度的平方根除以序列的元素总和。

下面来看另一个例子：1，0，0，0，0，1，0，0，0，0，1，0，0，0，0，1。这个序列有16个元素。直觉告诉我们周期大约为5，而频率稍微大于3。图17.5显示了它的傅立叶变换的绝对值。峰值再一次被分割成大约3部分，因此可以说频率大约为3。原始序列的周期大约为5，这与我们的直觉是一致的。

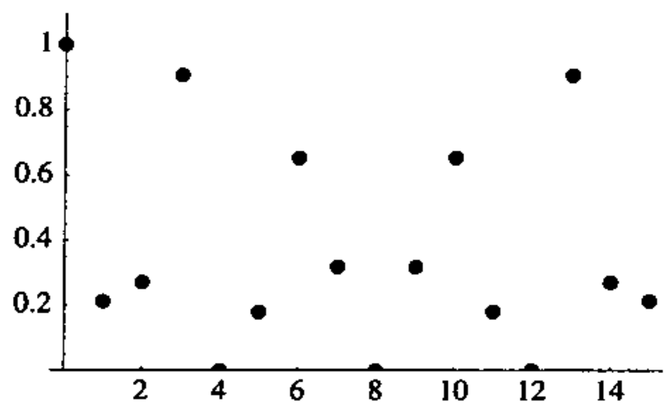


图17.5 离散傅立叶变换的绝对值

在前两个例子中，周期是序列长度（也就是8）的约数。我们仅仅在频率的倍数处获取傅立叶变换的非零值。在后两个例子中，周期不是序列长度（8或16）的约数。因为这种情形引入了一些“噪声”。在接近频率的倍数的点处得到峰值，而远离这些峰值的点的值趋近于0。

综上所述，傅立叶变换的峰值确定了序列的近似频率和周期。在 Shor 的算法中这将是非常有用的。

17.3.3　Shor 的算法

选择 m 使之满足 $n^2 \leqslant 2^m < 2n^2$。首先从 m 量子位开始，所有的都处于状态 0：

$$|000000000\rangle。$$

如前几节那样，通过改变轴可以将第一位转变成 $|0\rangle$ 和 $|1\rangle$ 的线性组合：

$$\frac{1}{\sqrt{2}}\ (|000000000\rangle + |100000000\rangle)。$$

然后对第 2 位、第 3 位、直到第 m 位连续地做相似的变换，得到量子状态：

$$\frac{1}{\sqrt{2^m}}(|000000000\rangle + |000000001\rangle + |000000010\rangle + \cdots + |111111111\rangle)。$$

因而 m 量子位的所有可能状态都被添加到这个和中。为了使符号简化，我们用等值的十进制数来替代 0 串和 1 串中的每一个串，所以可得到：

$$\frac{1}{\sqrt{2^m}}(|0\rangle + |1\rangle + |2\rangle + \cdots + |2^m - 1\rangle)。$$

选择一个满足 $1 < a < n$ 的随机数 a。可以假设 $\gcd(a,n) = 1$；否则可得到 n 的一个因子。为了得到量子的状态，量子计算机计算函数 $f(x) = a^x \pmod n$ 来获取

$$\frac{1}{\sqrt{2^m}}(|0,a^0\rangle + |1,a^1\rangle + |2,a^2\rangle + \cdots + |2^m - 1,a^{2^m-1}\rangle)$$

（由于符号的灵活性，a^x 被用来定义 $a^x \pmod n$）。这就给出了所有 a^x 值的列表。虽然这样，到目前为止并没有比用常规计算机好多少。如果测量系统的状态，就可以为某个随机选择的 x_0 得到一个基本状态 $|x_0,\ a^{x_0}\rangle$。甚至不能指定我们想用哪一个 x_0。此外，系统被迫使进入这种状态，并且删除了计算出来的所有其他 a^x 的值。因此，我们没有必要去测量整个系统。取而代之，我们仅测量后半部分的值。系统的每一个基本片都是 $|x,\ a^x\rangle$ 这样的形式，其中 x 表示 m 位，a^x 用 $m/2$ 位（因为 $a^x \pmod n < n < 2^{m/2}$）表示。如果我们测量最后的 $m/2$ 位，可以得到某个数 $u \pmod n$，并且整个系统被迫使进入那些形式为 $|x,\ u\rangle$ 的状态的组合，这些形式满足条件 $a^x \equiv u \pmod n$：

$$\frac{1}{C}\sum_{\substack{0 \leqslant x < 2^m \\ a^x \equiv u \pmod n}} |x,u\rangle,$$

其中 C 是使向量长度为 1 的任何因子（事实上，C 是总和中元素数目的平方根）。

举例：在某种意义上，大概很值得举这样一个例子。n 赋值为 21（这个例子似乎有点简单，但它比目前量子计算机能够处理的还要复杂一些!），因为 $21^2 < 2^9 < 2 \cdot 21^2$，所以 m 为 9。令 $a = 11$，这样计算 $11^x \pmod{21}$ 的值可以得到

$$\frac{1}{\sqrt{512}}(|0,1\rangle + |1,11\rangle + |2,16\rangle + |3,8\rangle + |4,4\rangle + |5,2\rangle + |6,1\rangle + |7,11\rangle +$$

$$|8,16\rangle + |9,8\rangle + |10,4\rangle + |11,2\rangle + |12,1\rangle + |13,11\rangle + |14,16\rangle +$$

$$|15,8\rangle+|16,4\rangle+|17,2\rangle+|18,1\rangle+|19,11\rangle+|20,16\rangle+\cdots$$
$$+|508,4\rangle+|509,2\rangle+|510,1\rangle+|511,11\rangle)。$$

假设我们测量第二部分，并得到了 2。这意味着我们已经提取出 $|x,2\rangle$ 这种形式的所有元素，并得到

$$\frac{1}{\sqrt{85}}(|5,2\rangle+|11,2\rangle+|17,2\rangle+|23,2\rangle+\cdots+|497,2\rangle+|503,2\rangle+|509,2\rangle)。$$

因为第二部分不再需要，为了表示方便，我们丢弃它并得到

$$\frac{1}{\sqrt{85}}(|5\rangle+|11\rangle+|17\rangle+|23\rangle+\cdots+|497\rangle+|503\rangle+|509\rangle)。$$

如果现在测量这个系统，仅仅能够得到满足 $11^x\equiv 2\pmod{21}$ 的数字 x。这将是没用的。■

假设能够进行两次测量，那么就可得到两个数字 x 和 y，满足 $11^x\equiv 11^y\pmod{21}$，得出 $11^{x-y}\equiv 1\pmod{21}$。利用指数因数分解方法（见 6.4 节），这将给我们很好的机会去因数分解 21。然而，我们不能接受两个相互独立的测量方法。第一种方法使系统进入输出状态，而第二种方法仅仅给出与第一种相同的答案。

但并非所有的都没用。注意在我们的例子中，状态中的数字是以 6 为周期的。通常情况下，$a^x\pmod n$ 的值是具有周期性的，周期为满足 $a^r\equiv 1\pmod n$ 的值 r。所以假设我们能够进行一次产生这个周期的测量，那么就有一种能够满足 $a^r\equiv 1\pmod n$ 的情况，因此就能够通过前面 6.4 节中提到的方法对 n 因数分解。

量子傅立叶变换（quantum Fourier transform）正是我们所需要的手段。它能够测量频率，而通过频率又可以发现周期。如果 r 碰巧是 2^m 的约数，那么我们得到的频率就是基本频率 f_0 的倍数，并且 $rf_0=2^m$。一般情况下，r 不会是 2^m 的约数，所以会得到一些占优势的频率，它们近似等于基本频率 f_0 的倍数，即：$rf_0\approx 2^m$。看下面的例子。

量子傅立叶变换通过公式 $QFT(|x\rangle)=\frac{1}{\sqrt{2^m}}\sum_{c=0}^{2^m-1}e^{\frac{2\pi icx}{2^m}}|c\rangle$ 被定义在基本状态 $|x\rangle(0\leqslant x<2^m)$。通过线性变换，将它扩展到状态的线性组合：

$$QFT(a_1|x_1\rangle+\cdots+a_t|x_t\rangle)=a_1QFT(|x_1\rangle)+\cdots+a_tQFT(|x_t\rangle)。$$

因此能够把 QFT 应用到我们的量子状态中。

在本例中，计算

$$QFT\left(\frac{1}{\sqrt{85}}(|5\rangle+|11\rangle+|17\rangle+|23\rangle+\cdots+|497\rangle+|503\rangle+|509\rangle)\right)$$

得到一个关于某些数 $g(c)$ 的总和 $\frac{1}{\sqrt{85}}\sum_{c=0}^{511}g(c)|c\rangle$。

数 $g(c)$ 由式子

$$g(c)=\frac{1}{\sqrt{512}}\sum_{\substack{0\leqslant x<512\\ x\equiv 5\pmod 6}}e^{\frac{2\pi icx}{512}}$$

给出，它是序列

0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, ..., 0, 0, 0, 0, 0, 1, 0, 0

的离散傅立叶变换。因此，g 的绝对值的图表峰值应该与序列的频率相对应，大约为

$512/6 \approx 85$。图 17.6 中的图表是 $|g|$ 的表示图。

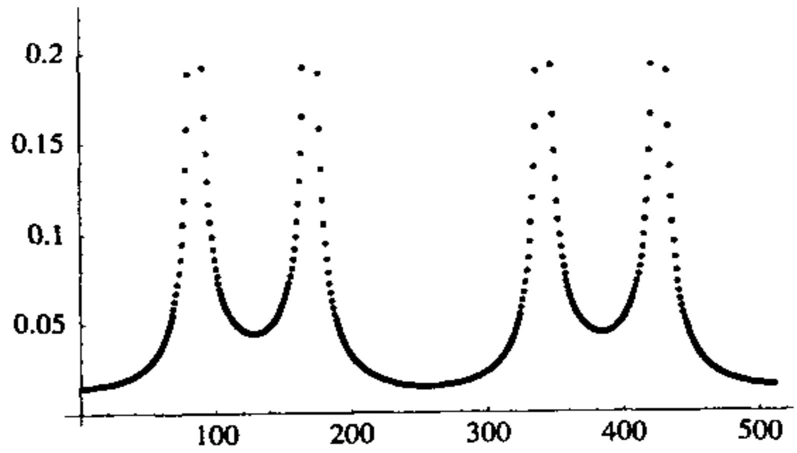


图 17.6 $g(c)$ 的绝对值

在 $c=0$，85，171，256，341，427（由于 0 和 256 两点的值被集中在一个值上，所以它们并没有显示在图表中；见下文）处有明显的峰值。这些是以前提到的占优势的频率。g 靠近 $c=341$ 处峰值的一些点的值为：

338	339	340	341	342	343	344	345
0.305	0.439	0.773	3.111	1.567	0.631	0.398	0.291

靠近 $c=85$，171 和 427 处点的情形与之类似。在 $c=0$ 和 256 处，可得到 $g(0)=3.756$，而所有 c 附近的值都满足 $g(c) \approx 0.015$。

在基本频率 $f_0=85$ 的倍数处峰值是类似的。当然，对此还没有真正了解，因为还没有进行任何测量。

现在我们测量这个傅立叶变换的量子状态。回想前面所提到的，如果从规格化为 $\sum |a_j|^2 = 1$ 的状态的线性组合 $a_1|x_1\rangle + \cdots + a_t|x_t\rangle$ 开始，那么获取 $|x_k\rangle$ 的概率为 $|a_k|^2$。更一般的情况下，如果我们不假设 $\sum |a_j|^2 = 1$，那么概率为

$$|a_k|^2 / \sum |a_j|^2。$$

在本例中，

$$3.111^2 / \sum |a_j|^2 \approx .114,$$

所以如果我们对傅立叶变换进行采样，那么得到 $c=85$，171，341，427 中之一的概率约为 $4 \times .114 = .456$。下面来假设一种情形，得到的是 $c=427$。我们知道，至少可以预计到，427 近似为频率 f_0 的倍数，即：

$$427 \approx jf_0,$$

其中 j 正是我们想得到的。因为 $rf_0 \approx 2^m = 512$，所以将两者相除得到

$$\frac{427}{512} \approx \frac{j}{r}。$$

注意，$427/512 \approx .834 \approx 5/6$。又因为必须使 r 满足 $r \leqslant \phi(21) < 21$，所以一个合理的猜测就是 $r=6$（看下面的连分数的讨论）。

一般来说，Shor 说明了对某个 j 有较高的机会得到满足条件

$$\left|\frac{c}{2^m}-\frac{j}{r}\right| < \frac{1}{2^{m+1}} < \frac{1}{2n^2}$$

的 $c/2^m$ 的值。连分数的方法将找到满足不等式 $r<n$ 的惟一（看练习 3）的 j/r 的值。

在本例子，我们接受 r 为 6，并对 $a^r = 11^6 \equiv 1 \pmod{21}$ 进行了核对。

我们想采用 6.4 节中指数因数分解的方法来因数分解 21。记得这种方法是令 $r=2^k m$，其中 m 为奇数，再计算 $b_0 \equiv a^m \pmod n$，然后就可以连续地自乘 b_0 来得到 b_1，b_2，…，直到达到 $1 \pmod n$。如果 b_u 是最后一个满足 $b_i \not\equiv 1 \pmod n$ 的值，那么我们就可以通过计算 $\gcd(b_u-1,n)$ 得到 n 的因子（可能没有什么价值）。

在本例中，令 $6=2\cdot 3$（一个奇数的两倍的幂），计算（6.4 节中的符号）

$$b_0 \equiv 11^3 \equiv 8 \pmod{21}$$

$$b_1 \equiv 11^6 \equiv 1 \pmod{21}$$

$$\gcd(b_0-1,21)=\gcd(7,21)=7,$$

得到 $21=7\cdot 3$。

一般地，一旦对于 r 有了一个候选值，我们就对 $a^r \equiv 1 \pmod n$ 进行核对。如果不满足，很遗憾，必须从一个新的值 a 开始，形成一个新的量子状态序列。如果 $a^r \equiv 1 \pmod n$，那么就可以采用 6.4 节的指数因数分解方法。如果对 n 进行因数分解失败，就从一个新的 a 值重新开始。经过多次尝试，n 的因数分解很有可能被发现。

17.3.4 连分数

最后我们说明一下怎样用连分数的方法找到分数 j/r。已知 $r\leq\phi(n)<n$，所以尽量用一个满足 $r<n$ 的有理数 j/r 来近似某个数（如 427/512）。

首先，考虑一下如何找到一个含有接近实数 x 的小分母的有理数的难题。例如，假设我们想近似 π。当然，可以用 314/100 = 157/50，但是通过使用含有更小的分母的 22/7 我们可以更加精确，所以难题在于采用比某些限度更小的分母获取最佳近似值。

对实数 x 来说大体的过程如下。令 $[x]$ 表示小于等于 x 的最大整数，令 $a_0=[x]$，$x_0=x$，然后定义

$$x_{i+1}=\frac{1}{x_i-a_i},\ a_{i+1}=[x_{i+1}]。$$

下面就是对于近似 π 怎样继续进行的步骤。令 $[\pi]=3$，则 $x_1=1/(\pi-3)\approx 7.06251$，$a_1=[x_1]=7$。再取 $x_2=1/(x_1-a_1)\approx 15.9966$，$a_2=15$。继续下去，可以得到 $x_3=1/(x_2-a_2)\approx 1.00342$，$a_3=1$，$x_4=1/(x_3-a_3)\approx 292.6$，$a_4=292$。这样继续扩展下去可得

$$\pi = 3+\cfrac{1}{7+\cfrac{1}{15+\cfrac{1}{1+\cfrac{1}{292+\cdots}}}}。$$

如果在经过了这个连分数的几层后停下来，可得到近似值

$$3, 3+\frac{1}{7}=\frac{22}{7}, \frac{333}{106}, \frac{355}{113}, \cdots。$$

最后的近似值是非常精确的：

$$\pi = 3.14159265\ldots,\text{而 } 355/113 = 3.14159292\ldots。$$

这个过程可以应用到任意实数 x 中，以产生一个有理数序列 r_1/s_1，r_2/s_2，…。产生的每一个有理数 r_k/s_k 给出了一个更精确的 x 的近似值，这个近似值比任何分母小于下一个分母 s_{k+1} 的有理数都精确。例如，在所有分母小于 106 的有理数中，22/7 是 π 的最精确的近似值。

现在让我们把这个过程应用到 427/512 上。可得到

$$\frac{427}{512}=0+\cfrac{1}{1+\cfrac{1}{5+\cfrac{1}{4+\cfrac{1}{2}}}}。$$

这产生了以下数值：

$$0,\ 1,\ \frac{5}{6},\ \frac{211}{253},\ \frac{427}{512}。$$

既然已知周期小于 $n=21$，那么最佳的推测是最后的分母小于 n，也就是 $r=6$。

一般情况下，我们计算连分数的 $c/2^m$ 层扩展，其中 c 是测量的结果。然后和以前一样计算近似值，最后小于 n 的分母就是 r 的候选数。

17.3.5　结束语

量子计算机和量子算法的性能对于经济和政府机构有着重大的意义。目前有许多秘密被加密协议所保护。最近几年里量子密码体制破解这些秘密的潜力和保护将来秘密的潜力已经导致了这个研究领域的飞速发展。

虽然第一台完全的量子计算机可能还要许多年才能完成，而且仍然有许多人对它的能力表示怀疑，但是量子密码体制已经成功运用在超过 24 千米的距离内传送可靠信息，并且能够处理量子位的（非常）小的数的量子计算机已经建立起来了。量子计算和密码体制已经改变了计算机科学家和工程师以前认识到的计算机的能力和限制。量子计算已经迅速发展成一个受欢迎的跨学科研究领域，并且有望在将来提供许多令人兴奋的新成就。

17.4　习　　题

1. 考虑序列 2^0，2^1，2^2，…（mod 15）。

(a) 这个序列的周期是多少？

(b) 假设要用 Shor 的算法来因数分解 n，n 为 15，那么 m 应取什么值？

(c) 假设通过 Shor 的算法的测量结果得到 $c=192$，那么得到的 r 值为多少？它与（a）一致吗？

(d) 用（c）得出的 r 值因数分解 15。

2. (a) 令 $0<s\leqslant m$，选定一个整数 c_0，使之满足 $0\leqslant c_0<2^s$。如果 $x\not\equiv 0(\bmod 2^{m-s})$，并且 $x=2^{m-s}$或者 $x\equiv 0$ (mod 2^{m-s})，证明

$$\sum_{\substack{0\leqslant c<2^m\\ c\equiv c_0\ (\bmod 2^s)}} e^{\frac{2\pi icx}{2^m}}=0。$$

(提示：令 $c=c_0+j2^s$，其中 $0\leqslant j<2^{m-s}$，从总和中因数分解出 $e^{2\pi ic_0x/2^m}$，辨别什么被作为一个几何和剩下。)

(b) 假设 a_0，a_1，...，a_{2^m-1}是一个长度为 2^m的序列，其中对于所有的 j，k 满足 $a_k=a_{k+j2^s}$。证明在满足 $x\not\equiv 0(\bmod 2^{m-s})$ 的情况下，序列的傅立叶变换 F (x) 为 0。

这说明了如果序列的周期是 2^m的约数，那么 F 的所有非零值产生在频率（也就是 2^{m-s}）的倍数上。

3. (a) 假设 j/r 和 j_1/r_1是两个完全不同的有理数，并且 $0<r<n$，$0<r_1<n$，证明

$$\left|\frac{j_1}{r_1}-\frac{j}{r}\right|>\frac{1}{n^2}。$$

(b) 如 Shor 的算法那样，假设已知

$$\left|\frac{c}{2^m}-\frac{j}{r}\right|<\frac{1}{2n^2}\text{和}\left|\frac{c}{2^m}-\frac{j_r}{r_1}\right|<\frac{1}{2n^2}，$$

证明 $j/r=j_1/r_1$。

4. (a) 计算 12345/11111 的连分数，并与 3.1 节的 gcd (12345, 11111) 的计算相比较。

(b) 证明欧几里得运算法则中关于 gcd (a, b) 的商确实是出现在连分数 a/b 中的数字 a_0，a_1，...。

5. (a) 在计算$\sqrt{3}$和$\sqrt{7}$的连分数的几个步骤中，有没有注意到任何模式？（可以说明形式为 $a+b\sqrt{d}$的每一个无理数的连分数中 a_i的值最后变成周期性的，其中 a，b，d 为有理数，$d>0$。）

(b) 计算 e 的连分数扩展的几个步骤中，有没有注意到任何模式？（另一方面，π 的连分数扩展似乎相当随机。）

Mathematica 实例

附录 A

附录中所介绍的计算机实例是用 Mathematica 编写的。如果你可以得到 Mathematica ，建议你在计算机上试用一下；如果没有 Mathematica，仍然可以阅读这些例子。它们提供了本书中所涉及的若干概念的例子。如何开始 Mathematica，见 A.1 节 。关于 Mathematica 的命令内容可到如下网址下载 Mathematica 命令集：

http ://www.prenhall.com/washington

A.1 Mathematica 入门

1. 打开链接 *http://www.prenhall.com/washington*，下载关于 Mathematica 的记录集文件 math.nb。

2. 打开 Mathematica，在 Mathematica 窗口的顶端命令栏下，使用菜单项打开 math.nb（也许在下载时会自动地执行；它取决于你的计算机设置）。

3. 在 math.nb 的前台 ，单击（左键）命令栏上的 Kernel 按钮，弹出一个菜单，它的第一行为 Evaluation。移动箭头选中第一行，出现下一层子菜单，移动箭头到 Evaluate Notebook 并点击，这个程序将开始装载必需的函数，忽视任何关于拼写的警告，这是因为一些函数有相似的名字。

4. 单击命令栏顶端的 File 项，移动箭头到 New 并单击，一个新的记录区将出现在 math.nb 上方，当然 math.nb 的所有命令仍在继续工作。

5. 如果想要给新的记录区命名 ，使用 File 菜单下的 Save As...，然后以 .nb 为后缀保存。

6. 现在就准备使用 Mathematica 了，如果你想尝试一些简单的东西，键入 1 + 2 * 3 + 4 ^ 5 ，然后同时按下 Shift 和 Enter 键。或者，如果你的键盘有带 Enter 的数字小键盘，可能在键盘的右边上，你就可以按下 Enter（不用按 Shift）。结果 1031 就会出现（它是 $1+2\cdot3+4^5$的结果）。

7. 转到 A.3 节计算机的例子，尝试敲一些命令，运行结果应该和例子中的结果相同，记住要按住 Shift 和 Enter 键（或数字的 Enter）以使 Mathematica 运行结果。

8. 如果想要删除部分记录，移动箭头到窗口右边的蓝线内并且按下左键，通过单击命令栏顶部的 Edit 按钮，然后在出现的菜单中单击 Cut 按钮，就可以删除高亮显示的部分。

9. 要保存记录，可以单击命令栏上的 File 项，然后在随后出现的菜单中单击 Save 按钮。

10. 单击命令栏顶部的 File，在显示的菜单中单击 Print 按钮，就可以打印所要的记录（你将看见在第 4 步打开一个新记录区的好处；如果没有打开，那么在 math. nb 上所有的命令都会被打印）。

11. 如果使用的命令出错并得到一条警告消息，可以编辑这些命令并且按 Shift + Enter 再试一次，不需要重新键入所有的东西。

12. 如果一个程序运行了很长时间，可以通过选择 Kernal and Abort Evaluation 来停止它。如果不起作用，在计算机上通常会有 Off 按钮。

13. 通过顶端的命令栏查看其他可用的命令，例如，Format then Style 允许在高亮显示的部分上（通过选择右边的蓝条）改变字体类型。

14. 如果你正在寻求帮助或用一个命令完成一些功能，试用 Help 命令，Master Index 将给出很多有用的信息。注意 Mathematica 命令总是以大写字母开始的，而来自 math. nb 的命令是以小写字母开始的，并且在 Help Index 上找不到。

15. 使用一些数论和绘图的命令要求装载特殊的程序包（例如，第 3 章的例 7）。当记录集和网络站点连接时这些包将自动地被装载。如果单独地使用记录集中的命令，不要忘记装载这些程序包。标识所需要的程序包的一种方法是在 Master Index 上查找这些命令。

A.2 部分命令

以下是用于计算机实例中的部分 Mathematica 命令。这些命令以大写字母开始，例如 **EulerPhi**，是 Mathematica 的组成部分，以小写字母开始的命令（例如 **addell**）是专门用来文本书写的，并且在 *http：//www.prenhall.com/washington* 网站上的 Mathematica 记录集中可以找到。

addell [{x, y}, {u, v}, a, b, n]　在椭圆曲线 $y^2 = x^3 + ax + b \pmod n$ 上求点 $\{x,y\}$ 和 $\{u,v\}$ 之和。

affinecrypt [txt, m, n]　使用 $mx + n$ 对文本 txt 的仿射加密。

ChineseRemainderTheorem [{a, b, ...}, {m, n, ...}]　给出同余方程组 $x \equiv a \pmod m$，$x \equiv b \pmod n$，... 的一个解。

choose [txt, m, n]　列出与 $m \pmod n$ 同余的文本 txt 的特征。

coinc [txt, n]　比较 txt 和其位移 n 位后两者相匹配的个数。

corr [v] 向量 v 与字母频率向量移位 26 次后产生的点积。

EulerPhi [n]　计算 $\phi(n)$（n 取值不能太大）。

ExtendedGCD [m, n]　计算满足条件 $mx + ny = \gcd$ 的 m 和 n 的最大公约数 gcd。

FactorInteger [n]　因数分解 n。

frequency [txt]　列出 txt 中每个字母 a 到 z 出现的次数。

GCD [m, n]　是 m 和 n 的最大公约数 gcd。

Inverse [M]　求矩阵 M 的逆矩阵。

lfsr [c, k, n]　以向量 c 为系数的递归运算产生的 n 位序列，初始值由向量 k 给出。

lfsrlength [v, n]　测试向量 v 的位数是否是通过循环长度至多为 n 的递归运算产生。

lfsrsolve [v, n] 给出用递归方法产生的二进制向量 v 的关于长度 n 的一个猜测值，可以计算递归的系数。

Max [v] 向量 v 最大的元素。

Mod [a, n] $a \pmod n$ 的值。

multell [{x, y}, m, a, b, n] 计算点 $\{x,y\}$ 在椭圆曲线 $y^2 \equiv x^3 + ax + b \pmod n$ 上的 m 次方。

multsell [{x, y}, m, a, b, n] 列出椭圆曲线 $y^2 \equiv x^3 + ax + b \pmod n$ 上点 $\{x, y\}$ 的第一个 m 的倍数。

NextPrime [x] 给出下一个大于 x 的素数（必须装载 NumberTheoryFunctions 包）。

num2text0 [n] 将数字 n 转换成字母，每一对连续的数字最大为 25；a 是 00，z 是 25。

num2text [n] 将数字 n 转换成字母，每一对连续的数字最大为 26；空是 00，a 是 01，z 是 26。

PowerMod [a, b, n] 计算 $a^b \pmod n$。

PrimitiveRoot [p] 找出素数 p 的本原根。

shift [txt, n] 将 txt 移动 n 位。

txt2num0 [txt] 用 $a=00$, ..., $z=25$ 的形式将文本 txt 变换成数字。

txt2num [txt] 用空 $=00$, $a=01$, ..., $z=26$ 的形式将文本 txt 变换成数字。

vigenere [txt, v] 给出使用向量 v 加密 txt 的 Vigenère 密码。

vigvec [txt, m, n] 给出字母 a 到 z 在与 $n \pmod m$ 同余情况下的出现频率。

A.3 第 2 章实例

例 1：使用移位密码获得密文 **kddmu**，尝试所有的可能性来解密它。

In [1]:= **allshifts [" kddkmu"]**

```
kddkmu
leelnv
mffmow
nggnpx
ohhoqy
piiprz
qjjqsa
rkkrtb
sllsuc
tmmtvd
unnuwe
voovxf
wppwyg
xqqxzh
```

```
yrryai
zsszbj
attack
buubdl
cvvcem
dwwdfn
exxego
fyyfhp
gzzgiq
haahjr
ibbiks
jccjlt
```

可见，*attack* 是这些列表项中惟一的一个单词，显然这就是明文。

例 2：用仿射函数 $7x+8$ 对明文 *cleopatra* 加密：

In [2]:= **affinecrypt ["cleopatra", 7, 8]**

Out [2]:= whkcjilxi

例 3：密文 *mzdvezc* 是用仿射函数 $5x+12$ 加密的，试对它解密。

解：首先，解 $y \equiv 5x+12 \pmod{26}$ 得 $x \equiv 5^{-1}(y-12)$ 。我们需要找到 5（mod26）的逆元素：

In [3]:= **PowerMod [5, −1, 26]**

Out [3] = 21

因此，$x \equiv 21(y-12) \equiv 21y - 12 \cdot 21$。把 $-12 \cdot 21$ 转换为标准形式：

In [4]:= **Mod [−12 * 21, 26]**

Out [4] = 8

因此，解密函数是 $x \equiv 21y+8$，解密信息为：

In [5]:= **affinecrypt ["mzdvezc", 21, 8]**

Out [5]:= anthony

所以可以得出，明文是用如下形式加密的：

In [6]:= **affinecrypt ["anthony", 5, 12]**

Out [6] = mzdvezc

例 4：这是一个关于文本的 Vigenère 密码的例子，让我们看看怎么用 2.3 节的方法来破解密文，为了方便，我们已经存储了名为 *vvhq* 的加密文本。

In [7]:= **vvhq**

Out [7] =

```
vvhqwvvrhmusgjgthkihtssejchlsfcbgvwcrlryqtfsvgahwkcuhwauglq
hnslrljshbltspisprdxljsveeghlqwkasskuwepwqtwvspgoelkcqyfnsv
wljsniqkgnrgybwlwgoviokhkazkqkxzgyhcecmeiujoqkwfwvefqhkijrc
lrlkbienqfrjljsdhgrhlsfqtwlauqrhwdmwlgusgikkflryvcwvspgpmlk
assjvoqxeggveyggzmljcxxljsvpaivwikvrdrygfrjljslveggveyggeia
puuisfpbtgnwwmuczrvtwglrwugumnczvile
```

找出密文中字母的频率：

In [8]:= **frequency [vvhq]**

Out [8]=

{{a, 8}, {b, 5}, {c, 12}, {d, 4}, {e, 15}, {f, 10}, {g, 27}, {h, 16}, {i, 13},
{j, 14}, {k, 17}, {l, 25}, {m, 7}, {n, 7}, {o, 5}, {p, 9}, {q, 14}, {r, 17}, {s,
24},
{t, 8}, {u, 12}, {v, 22}, {w, 22}, {x, 5}, {y, 8}, {z, 5}}

让我们计算移动 1，2，3，4，5，6 位后相同的字母数：

In [9]:= **coinc [vvhq, 1]**

Out [9]=14

In [10]:= **coinc [vvhq, 2]**

Out [10]=14

In [11]:= **coinc [vvhq, 3]**

Out [11]=16

In [12]:= **coinc [vvhq, 4]**

Out [12]=14

In [13]:= **coinc [vvhq, 5]**

Out [13]=24

In [14]:= **coinc [vvhq, 6]**

Out [14]=12

我们得出结论：密钥的长度可能是 5 位，再来看看第 1 个，第 6 个，第 11 个，... 字母(即，这些字母的位置都是与 1 mod 5 同余的)：

In [15]:= **choose [vvhq, 5, 1]**

Out [15]=

vvuttcccqgcunjtpjgkuqpknjkygkkgcjfqrkqjrqudukvpkvggjjivgjggpfncwuce

In [16]:= **frequency [%]**

Out [16]= {{a, 0}, {b, 0}, {c, 7}, {d, 1}, {e, 1}, {f, 2}, {g, 9}, {h, 0}, {i,
1},
{j, 8}, {k, 8}, {l, 0}, {m, 0}, {n, 3}, {o, 0}, {p, 4}, {q, 5}, {r, 2}, {s, 0},
{t, 3}, {u, 6}, {v, 5}, {w, 1}, {x, 0}, {y, 1}, {z, 0}}

将其表示为一个频率向量为：

In [17]:= **vigvec [vvhq, 5, 1]**

Out [17]= {0, 0, 0.104478, 0.0149254, 0.0149254, 0.0298507, 0.134328, 0,
0.0149254, 0.119403, 0.119403, 0, 0, 0.0447761, 0, 0.0597015, 0.0746269,
0.0298507, 0, 0.0447761, 0.0895522, 0.0746269, 0.0149254, 0, 0.0149254, 0}

这个向量与字母频率向量移位后的点积计算如下：

In [18]:= **corr [%]**

Out [18]=

{0.0250149, 0.0391045, 0.0713284, 0.0388209, 0.0274925, 0.0380149, 0.051209,
0.0301493, 0.0324776, 0.0430299, 0.0337761, 0.0298507, 0.0342687, 0.0445672,

0.0355522, 0.0402239, 0.0434328, 0.0501791, 0.0391791, 0.0295821, 0.0326269, 0.0391791, 0.0365522, 0.0316119, 0.0488358, 0.0349403}

第三个位置是最大值，但有时最大的元素是很难确定的。确定它的一种方法是：

In [19]:= **Max [%]**

Out [19]=0.0713284

现在很容易从清单中看到并找出这个数（它通常只出现一次）。由于它在第三个位置出现，对这个 Vigenère 密码移 2 位，相应的字母为 *c*。这个程序和刚刚使用过的非常相似（使用 *vigvec* [*vvhq*, 5, 2], ..., *vigvec* [*vvhq*, 5, 5]），这表示其他的移位很可能是 14, 3, 4, 18。我们通过解密来验证一下它的正确性。

In [20]:= **vigenere [vvhq, - {2, 14, 3, 4, 18}]**

Out [20]=

themethodusedforthepreparationandreadingofcodemessagesissimpleintheextremean
datthesametimeimpossibleoftranslationunlessthekeyisknowntheeasewithwhichthek
eymaybechangedisanotherpointinfavoroftheadoptionofthiscodebythosedesiringtot
ransmitimportantmessageswithouttheslightestdangeroftheirmessagesbeingreadbyp
oliticalorbusinessrivalsetc

回顾一下，明文最初是用如下命令加密的：

In [21]:= **vigenere [%, {2, 14, 3, 4, 18}]**

Out [21]=

vvhqwvvrhmusgjgthkihtssejchlsfcbgvwcrlryqtfsvgahwkcuhwauglqhnslrjshbltspispr
dxljsveeghlqwkasskuwepwqtwvsqgoelkcqyfnsvwljsniqkgnrgybwlwgoviokhkazkqkxzgyh
cecmeiujoqkwfwvefqhkijrclrlkbienqfrjlfsdhgrhlsfqrwlauqrhwdmwlgusgikkflryvcwv
spgpmlkassjvoqxeggveyggzmljcxxljsvpaivwikvrdrygfrjlfslveggveyggeiapuuisfpbtg
nwwmuczrvtwglrwugumnczvile

例 5：密文

$$22,09,00,12,03,01,10,03,04,08,01,17$$

是通过矩阵

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 10 \end{pmatrix}。$$

使用希尔密码加密的，试对它解密。

解：把矩阵 $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ 作为 {{a,b},{c,d}} 输入。形式 *M.N* 指的是矩阵 *M* 和矩阵 *N* 相乘，形式 *v.M* 为向量 *v* 右乘矩阵 *M*。

首先，我们需要求矩阵对 26 取模的逆：

In [22]:= **Inverse [{{1, 2, 3}, {4, 5, 6}, {7, 8, 10}}]**

Out [22]= {{-2/3, -4/3, 1}, {2/3, 11/3, -2}, {1, -2, 1}}

由于是工作在模 26 方式下，我们不能让像 2/3 这样的分数作为结果。需去掉分母并减去 26 的模。这样做之后，把分数乘以 3 以提取出分子，然后乘 3 mod 26 的逆元素，置换出“分母”（见 3.3 节）：

In [23]:= % * 3

Out [23]= {{-2, -4, 3}, {-2, 11, -6}, {3, -6, 3}}

In [24]:= Mod [PowerMod [3, -1, 26] * %, 26]

Out [24]= {{8, 16, 1}, {8, 21, 24}, {1, 24, 1}}

这是矩阵对 26 取模后的逆。我们可验证如下：

In [25]:= Mod [%. {{1, 2, 3}, {4, 5, 6}, {7, 8, 10}}, 26]

Out [25]= {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}}

为了解码，我们把密文分成每 3 个字母一组，并且每一组右乘刚才计算出来的逆矩阵：

In [26]:= Mod [{22, 09, 00} . %%, 26]

Out [26]= {14, 21, 4}

In [27]:= Mod [{12, 03, 01} . %%%, 26]

Out [27]= {17, 19, 7}

In [28]:= Mod [{10, 03, 04} . %%%%, 26]

Out [28]= {4, 7, 8}

In [29]:= Mod [{08, 01, 17} . %%%%%, 26]

Out [29]= {11, 11, 23}

因此，明文为 14，21，4，17，19，7，4，7，8，11，11，23。可以转回成字母：

In [30]:= alph0 [142104171907040708111123]

Out [30]= overthehillx

注意在明文最后加的那个 x 是为了凑成 3 个字母的分组。

例 6：计算递归式子 $x_{n+5} \equiv x_n + x_{n+2} \pmod 2$ 的前 50 项。初始值为 0，1，0，0，0。

解：系数向量是 {1，0，1，0，0}，初始向量是 {0，1，0，0，0}，于是有

In [31]:= lfsr [{1, 0, 1, 0, 0}, {0, 1, 0, 0, 0}, 50]

Out [31]= {0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1}

例 7：假设 LFSR 序列的前 20 项是 1，0，1，0，1，1，1，0，0，0，0，1，1，1，0，1，0，1，0，1，找出生成这个序列的递归式。

解：首先，我们确定递归运算的长度。*lfsrlength [v, n]* 命令用于计算在 2.11 节中出现的前 n 个阵列的模 2 行列式。

In [32]:=

lfsrlength [{1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1}, 10]

{1, 1}

{2, 1}

{3, 0}

{4, 1}

{5, 0}

{6, 1}

{7, 0}

{8, 0}

{9, 0}

{10, 0}

最后的一个非零行列式是第六项，所以我们猜到递归的长度为6。就可以确定系数为：

In [33]:= **lfsrsolve [{1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1}, 6]**

Out [33]= {1, 0, 1, 1, 1, 0}

得到递归式如下

$$x_{n+6} \equiv x_n + x_{n+2} + x_{n+3} + x_{n+4} \pmod 2)。$$

例8：密文0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0是由明文对2取模再加上LFSR的输出得到的（即，用明文同LFSR的输出进行异或）。假设知道明文是以1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0开始的，求剩下的明文。

解：用已知的部分明文与密文进行异或运算，以获得LFSR输出的开始部分：

In [34]:= **Mod [{1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0} + {0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1}, 2]**

Out [34]= {1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1}

这就是LFSR输出的开始部分。下面我们求递归的长度：

In [35]:= **lfsrlength [%, 8]**

{1, 1}

{2, 0}

{3, 1}

{4, 0}

{5, 1}

{6, 0}

{7, 0}

{8, 0}

我们猜它的长度为5，为找出递归的系数：

In [36]:= **lfsrsolve [%%, 5]**

Out [36]= {1, 1, 0, 0, 1}

现在我们使用刚刚得出的系数再加上LFSR输出的前5项可以得出LFSR的完全输出：

In [37]:= **lfsr [{1, 1, 0, 0, 1}, {1, 0, 0, 1, 0}, 40]**

Out [37]= {1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0}

当用密文与LFSR输出进行异或，就可以得到明文：

In [38]:= **Mod [% + {0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0}, 2]**

Out [38]= {1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0}

这样就得出了明文。

A.4 第3章实例

例 1：找出 gcd（23456，987654）。

In [1]:= **GCD [23456, 987654]**

Out [1] = 2

例 2：在整数范围内求 x，y，使之满足 $23456x + 987654y = 2$。

In [2]:= **ExtendedGCD [23456, 987654]**

Out [2] = {2, {-3158, 75}}

这就意味着 2 是最大公约数，并且 $23456 \cdot (-3158) + 987654 \cdot 75 = 2$。

例 3：计算 $234 \cdot 456 \pmod{789}$。

In [3]:= **Mod [234 * 456, 789]**

Out [3] = 189

例 4：计算 $234567^{876543} \pmod{565656565}$。

In [4]:= **PowerMod [234567, 876543, 565656565]**

Out [4] = 473011223

例 5：求 87878787（mod 9191919191）的乘法逆元素。

In [5]:= **PowerMod [87878787, -1, 9191919191]**

Out [5] = 7079995354

例 6：求解 $7654x \equiv 2389 \pmod{65537}$。

解：方法一：

In [6]:=

Solve [{7654 * x == 2389, Modulus == 65537}, x,

Mode- > Modular]

Out [6] = {{Modulus - > 65537, x - > 43626}}

方法二：它与 3.3 节中的方法相对应。我们先计算 7654^{-1}，然后再乘以 2389；

In [7]:= **PowerMod [7654, -1, 65537]**

Out [7] = 54637

In [8]:= **Mod [% * 2389, 65537]**

Out [8] = 43626

例 7：找出满足下列条件的 x：

$$x \equiv 2 \pmod{78}, x \equiv 5 \pmod{97}, x \equiv 1 \pmod{119}。$$

解：首先，我们需要导入一些数字公理包：

In [9]:= < < **NumberTheory ' NumberTheoryFunctions '**

现在来求解这个问题：

In [10]: = **ChineseRemainderTheorem [{2, 5, 1}, {78, 97, 119}]**

Out [10] = 647480

检验结果：

In [11]:= **Mod [647480, {78, 97, 119}]**

Out [11]= {2, 5, 1}

例 8：把 123450 因数分解为素数。

In [12]:= **FactorInteger [123450]**

Out [12]= {{2, 1}, {3, 1}, {5, 2}, {823, 1}}

即 $123450 = 2^1 3^1 5^2 823^1$。

例 9：求 $\phi(12345)$ 的值。

In [13]:= **EulerPhi [12345]**

Out [13]= 6576

例 10：找出素数 65537 的本原根。

解：本题也需要导入一些数字公理包（这个过程前面已做过），然后键入

In [14]:= **PrimitiveRoot [65537]**

Out [14]= 3

因此，3 就是素数 65537 的本原根。

例 11：找出矩阵 $\begin{pmatrix} 13 & 12 & 35 \\ 41 & 53 & 62 \\ 71 & 68 & 10 \end{pmatrix}$ (mod 999) 的逆矩阵。

解：首先转换不带模的矩阵：

In [15]:= **Inverse [{{13, 12, 35}, {41, 53, 62}, {71, 68, 10}}]**

Out [15]= $\left\{\left\{\frac{3686}{34139}, -\frac{2260}{34139}, -\frac{1111}{34139}\right\}, \left\{-\frac{3992}{34139}, \frac{2355}{34139}, -\frac{629}{34139}\right\}, \left\{\frac{975}{34139}, \frac{32}{34139}, -\frac{197}{34139}\right\}\right\}$

现在我们需要除去分母 34139，于是计算 1/34139 mod 999：

In [16]:= **PowerMod [34139, −1, 999]**

Out [16]= 410

由于 $410 \cdot 34139 \equiv 1 \pmod{999}$，用 $410 \cdot 34139$ 乘以逆矩阵，并减去模 999，以便在不改变任何（mod 999）的情况下除去分母：

In [17]:= **Mod [410 * 34139 * %%, 999]**

Out [17]= {{772, 472, 965}, {641, 516, 851}, {150, 133, 149}}

因此，逆矩阵对 999 取模为 $\begin{pmatrix} 772 & 472 & 965 \\ 641 & 516 & 851 \\ 150 & 133 & 149 \end{pmatrix}$。

一般情况下，通过检查必须消去的公分母是可以确定结果的。如果不行，通常是把原始矩阵的行列式作为公分母。

例 12：求 26951623672 模素数 $p = 98573007539$ 的平方根。

解：由于 $p \equiv 3 \pmod 4$，我们可以使用 3.9 节中的方法：

In [18]:= **PowerMod [26951623672, (98573007539 + 1)/4, 98573007539]**

Out [18] = 98338017685

另一个平方根是它的负数：

In [19]:= **Mod [- %, 98573007539]**

Out [19] = 234989854

例 13：令 $n = 34222273 = 9803 \cdot 3491$，求 $x^2 \equiv 19101358 \pmod{34222273}$ 的所有 4 个解。

解：首先，找一个平方根分别对两个素数因子取模，两者都与 3（mod 4）同余：

In [20]:= **PowerMod [19101358, (9803 + 1)/4, 9803]**

Out [20] = 3998

In [21]:= **PowerMod [19101358, (3491 + 1)/4, 3491]**

Out [21] = 1318

因此，这个平方根与 ± 3998（mod 9803）和 ± 1318（mod 3491）是同余的。利用中国剩余定理，有 4 种方法来组合这些：

In [22]:= **ChineseRemainderTheorem [{3998, 1318}, {9803, 3491}]**

Out [22] = 43210

In [23]:= **ChineseRemainderTheorem [{ -3998, 1318}, {9803, 3491}]**

Out [23] = 8397173

In [24]:= **ChineseRemainderTheorem [{3998, -1318}, {9803, 3491}]**

Out [24] = 25825100

In [25]:= **ChineseRemainderTheorem [{ -3998, -1318}, {9803, 3491}]**

Out [25] = 34179063

这就是所要求的 4 个平方根。

A.5　第 6 章实例

例 1：假如需要从 50 个数中得到一个最大的随机素数。可以这样来做，首先，装载数字公理包：

In [1]:= **<<NumberTheory 'NumberTheoryFunctions '**

函数 *NextPrime* [*x*] 寻找比 *x* 大的下一个素数。函数 *Random* [*Integer*, {*a*, *b*}] 给出 a 到 b 之间的一个随机整数。综合以上，我们能得到一个素数：

In [2]:= **NextPrime [Random [Integer, {10^49, 10^50}]]**

Out [2] = 73050570031667109175215303340488313456708913284291

如果重复这个过程，我们能得到另一个素数：

In [3]:= **NextPrime [Random [Integer, {10^49, 10^50}]]**

Out [3] = 97476407694931303255724326040586144145341054568331

例 2：假设想把文本 *hellohowareyou* 转换成数字：

In [4]:= **num1 [" hellohowareyou"]**

Out [4] = 8051212150815230118052515 21

注意：使用 $a=1$，$b=2$，...，$z=26$，否则 a 在信息首部时将消失（一个更高效的方法建立在基数 27 之上，即信息的数字形式为 $8+5\cdot27+12\cdot27^2+\cdots+21\cdot27^{13}=87495221502384554951$，注意这种方法使用更少的位数）。

假如想把它转换回字母：

In [5]:= **alph1 [805121215081523011805251521]**

Out [5] = hellohowareyou

例 3：在 $n=823091$，$e=17$ 的条件下用 RSA 法加密信息 *hi*。

解：首先，把信息转换成数字：

In [6]:= **num1 [" hi"]**

Out [6] = 809

现在，把它提升到 e 次幂对 n 取模：

In [7]:= **PowerMod [%, 17, 823091]**

Out [7] = 596912

例 4：对上例的密文解密。

解：首先，我们需要得到解密指数 d。为此，要找到 ϕ (823091)。一种方法是：

In [8]:= **EulerPhi [823091]**

Out [8] = 821184

另一种方法是把 n 分解成 $p\cdot q$ 的形式，然后计算 $(p-1)(q-1)$：

In [9]:= **FactorInteger [823091]**

Out [9] = {{659, 1}, {1249, 1}}

In [10]:= **658 * 1248**

Out [10] = 821184

由 $de\equiv1(\bmod\phi(n))$，有如下计算（注意，我们寻找 $e \bmod \phi(n)$ 的逆，而不是 $\bmod n$ 的逆）：

In [11]:= **PowerMod [17, -1, 821184]**

Out [11] = 48305

因此，$d=48305$。为了解密，提升密文到 d 次方对 n 取模：

In [12]:= **PowerMod [596912, 48305, 823091]**

Out [12] = 809

最后，转换回字母：

In [13]:= **alph1 [809]**

Out [13] = hi

例 5：当 $n=823091$ 和 $e=17$ 时使用 RSA 体制加密信息 *hellohowareyou*。

解：首先，把明文转换成数字：

In [14]:= **num1 [" hellohowareyou"]**

Out [14] = 805121215081523011805251521

假设我们只简单地提升到 e 次幂对 n 取模：

In [15]:= **PowerMod [%, 17, 823091]**

Out [15] = 447613

如果解密（从例 4 可以求出 d），即得

In [16]: = **PowerMod [%, 48305, 823091]**

Out [16] = 628883

这不是原来的明文。原因是明文比 n 大，因此我们已经得到了明文模 n：

In [17]:= **Mod [805121215081523011805251521, 823091]**

Out [17] = 628883

我们需要把明文分组，每组小于 n，在本例中，我们一次使用 3 个字母：

80512 121508 152301 180525 1521

In [18]:= **PowerMod [80512, 17, 823091]**

Out [18] = 757396

In [19]:= **PowerMod [121508, 17, 823091]**

Out [19] = 164513

In [20]:= **PowerMod [152301, 17, 823091]**

Out [20] = 121217

In [21]:= **PowerMod [180525, 17, 823091]**

Out [21] = 594220

In [22]:= **PowerMod [1521, 17, 823091]**

Out [22] = 442163

因此密文为 757396164513121217594220442163。注意：这个密文无法还原回字母。实际上，它不与任何字母文本相对应。

分别解密每一组：

In [23]:= **PowerMod [757396, 48305, 823091]**

Out [23] = 80512

In [24]:= **PowerMod [164513, 48305, 823091]**

Out [24] = 121508

等等。

下面我们举一些大数字的例子，即在 6.5 节中讨论的关于 RSA 的挑战中所涉及的例子。它们存储在名为 *rsan*，*rsae*，*rsap*，*rsaq* 的文件下。

In [25]:= rsan

Out [25] =

114381625757888867669235779976146612010218296721242362562561842935706935245733897830597123563958705058989075147599290026879543541

In [26]:= **rsae**

Out [26] = 9007

例 6：使用 *rsan* 和 *rsae* 分别对 *b*，*ba*，*bar*，*bard* 加密。

In [27]:= **PowerMod [num1 [" b"], rsae, rsan]**

Out [27] =

7094675846761266859837016499155078618287633106068523541056470411448678226171649720012215533234846201405328798758089926376514253 4

In [28]:= **PowerMod [num1 [" ba"], rsae, rsan]**

Out [28] =

3504513060897510032501170944987195427378820475394859306031369769822762175980602796227053803156556477335203367178226130579615895 1

In [29]:= **PowerMod [num1 [" bar"], rsae, rsan]**

Out [29] =

4481451286385510107600453085949210934242953160660740907036054340800084364598688040595310281831282258636258029878444115192260642 4

In [30]:= **PowerMod [num1 [" bard"], rsae, rsan]**

Out [30] =

2423807778511166642320286251209031739348521295905627078313499161425605432329717980492895807344575266302644987398687798932990949 8

观察到这些密文的长度是相同的，这使得我们很难有一种简单的方法确定出相应明文的长度。

例 7：利用因数分解 *rsan* = *rsap* · *rsaq*，求对于 RSA 挑战的解密指数，并解密这个原文（见 6.5 节）。

解：首先找到解密指数：

In [31]:= **rsad = PowerMod [rsae, −1, (rsap −1) * (rsaq −1)];**

注：我们使用分号避免打印出结果。如果想查看 *rsad* 的数值，见 6.5 节或不用分号。要对以文件名为 *rsaci* 存储的密文解密，转换成字母即可：

In [32]:= **alph1 [PowerMod [rsaci, rsad, rsan]]**

Out [32] =

the magic words are squeamish ossifrage

例 8：使用 *rsan* 和 *rsae* 对信息 *rsaencryptsmessageswell* 加密。

In [33]:= **PowerMod [num1 [" rsaencryptsmessageswell"], rsae, rsan]**

Out [33] =

9463942034900225931630582353924949641464096993400170972140435241827195065425436558490601396632881775353928311265319755313078188 4

例 9：对上一题的密文解密。

解：幸好我们知道解密指数 *rsad*，因此，计算

In [34]:= **PowerMod [%, rsad, rsan]**

Out [34] = 1819010514031825162019130519190107051923051212

In [35]:= **alph1 [%]**

Out [35] = rsaencryptsmessageswell

假设我们在传输的时候丢失了密文的最后 4 位。试着对剩下的密文解密（减去 4 并除以 10 就可以将后 4 位移去）：

In [36]:= **PowerMod [(%%% −4) /10, rsad, rsan]**

Out [36] =

47952999173195988664902352629525486409113633894375629846854907970588412300373487969657794254117158956921267912628461494475682806

如果试着把它转换为字母，会得到一长串的错误信息；通常明文中的一个小错误就能完全地改变解密信息并产生毫无意义的数据。

例 10：已知 $n = 11313771275590312567$ 是两个素数的乘积，$\phi(n) = 11313771187608744400$。因数分解 n。

解：我们知道（见 6.1 节）p、q 为方程 $X^2-(n-\phi(n)+1)X+n$ 的两个根。因此，计算

In [37]:= **Roots [X^2 − (1131377127559031256 − 11313771187608744400 + 1) * X + 11313771275590312567 == 0, X]**

Out [37] = X == 128781017 || X == 87852787151

因此，$n = 128781017 \cdot 87852787151$。我们也可以用二次配方法来求它的根。

例 11：设已知 *rsae* 和 *rsad*，使用这些因数分解 *rsan*。

解：使用 6.4 节的通用指数因数分解方法。首先，记 $rsae \cdot rsad - 1 = 2^s m$，$m$ 是奇数。一种方法是先计算 $rsae \cdot rsad - 1$，然后使它除以 2 直到得到一个奇数：

In [38]:= **rsae * rsad − 1**

Out [38] =

961034419617782266156919023359583834109854129051878330250644604041155985575087352659156174898557342995131594680431086921245830097664

In [39]:= **%/2**

Out [39] =

480517209808891133078459511679791917054927064525939165125322302020577992787543676329578087449278671497565797340215543460622915048832

In [40]:= **%/2**

Out [40] =

240258604904445566539229755839895958527463532262969582562661151010288996393771838164789043724639335748782898670107771730311457524416

我们一直这样计算六步直到得到：

Out [46] =

3754040701631961977175464934998374351991617691608899727541580484535765568652684971324828808197489621074732791720433933286116523819

这个数是 m。现在选择一个随机整数 a，假设选到的是 13，利用通用指数因数分解法，计算：

In [47]:= **PowerMod [13, %, rsan]**

Out [47] =

27574368507006530592243494868847161198423085707307805690569839647030183109839862370800529338092984795490192643587960859870551239

要是不等于 $\pm 1 \pmod{rsan}$，我们就继续平方它直到得到 ±1：

In [48]:= **PowerMod [%, 2, rsan]**

Out [48] =

4831896032192851558013847641872303455410409906994084622549470277665499641258
2955636035266156108686431194298574075854037512277292

In [49]:= **PowerMod [%, 2, rsan]**

Out [49] =

7817281415487735657914192805875400002194878705648382091793062511521518183974
2056013275521913487560944732073516487722273875579363

In [50]:= **PowerMod [%, 2, rsan]**

Out [50] =

4283619120250872874219929904058290020297622291601776716755187021650944451823
9462186379470569442055101392992293082259601738228702

In [51]:= **PowerMod [%, 2, rsan]**

Out [51] = 1

由于 1 以前的最后一位数不是 ±1 (mod *rsan*)，我们有一个当 $x^2 \equiv 1$ 时 $x \not\equiv \pm 1 \pmod{rsan}$ 的例子。因此，gcd($x-1$, *rsan*)是 *rsan* 的一个非平凡因子：

In [52]:= **GCD [%% −1, rsan]**

Out [52] =

32769132993266709549961988190834461413177642967992942539798288533

这就是 *rsaq*。另一个因子通过计算 *rsan*/*rsaq* 可得到：

In [53]:= **rsan/%**

Out [53] =

3490529510847650949147849619903898133417764638493387843990820577

这是 *rsap*。

例 12：假设已知 $150883475569451^2 \equiv 16887570532858^2 \pmod{205611444308117}$，因数分解 205611444308117。

解：利用 6.3 节的基本原理。

In [54]:= **GCD [150883475569451 − 16887570532858, 205611444308117]**

Out [54] = 23495881

这就给出了一个因子，另一个因子是：

In [55]:= **205611444308117/%**

Out [55] = 8750957

我们能够检验这些因子是素数，所以不能再进一步因数分解了：

In [56]:= **PrimeQ [%%]**

Out [56] = Ture

In [57]:= PrimeQ [%%]

Out [57] = True

例 13：用 $p-1$ 的方法因数分解 n = 376875575426394855599989992897873239。

解：选择 B = 100 为界，并设 a = 2，于是计算 $2^{100!} \pmod{n}$：

In [58]:= **PowerMod [2, Factorial [100], *376875575426394855599989992897873239*]**

Out [58] = 369676678301956331939422106251199512

然后我们计算 $2^{100!}-1$ 和 n 的 gcd:

In [59]:= **GCD [%-1, 376875575426394855599989992897873239]**

Out [59] = 430553161739796481

这就是因子 p，另一个因子 q 为:

In [60]:= **376875575426394855599989992897873239/%**

Out [60] = 875328783798732119

下面来看看为什么这样计算。$p-1$ 和 $q-1$ 的因数分解是:

In [61]:= **FactorInteger [430553161739796481 -1]**

Out [61] = {{2, 18}, {3, 7}, {5, 1}, {7, 4}, {11, 3}, {47, 1}}

In [62]:= **FactorInteger [875328783798732119 -1]**

Out [62] = {{2, 1}, {61, 1}, {20357, 1}, {39301, 1}, {8967967, 1}}

我们看到 100! 是 $p-1$ 的倍数，所以 $2^{100!}\equiv 1 \pmod p$。然而，100! 不是 $q-1$ 的倍数，所以很可能 $2^{100!}\not\equiv 1 \pmod q$。因此，$2^{100!}-1$ 和 pq 都有一个相同的因子 p，但只有 pq 以 q 为因子。可见最大公约数 gcd 是 p。

A.6　第 8 章实例

例 1：设一个屋子里有 23 个人，至少有两个人生日相同的概率是多少?

解：没有两个人相同的概率是 $\prod_{i=1}^{22}(1-i/365)$（注意乘积是到 $i=22$ 而不是 $i=23$）。用 1 减去这个概率就是两个人生日相同的概率:

In [1]:= **1 - Product [1. - i/365, {i, 22}]**

Out [1] = 0.507297

注：在求积时用 1. 代替没有小数点的 1。如果去掉这个小数点，将是计算有理数的积（试试看，就知道了)。

例 2：设一个懒惰的电话公司职员用随机的 7 位数来分配电话号码。在一个有 10000 步电话的城市里，两个人分到同一号码的概率是多少?

In [2]:= **1 - Product [1. - i/10^7, {i, 9999}]**

Out [2] = 0.99327

注：电话号码大约是可能电话数量的平方根的 3 倍。这就意味着概率非常高。由 8.4 节，我们推测如果有 $\sqrt{2(\ln 2)10^7}\approx 3723$ 个左右的电话，就有 50% 的号码相同的机会。下面来验证一下它的正确性:

In [3]:= **1 - Product [1. - i/10^7, i, 3722]**

Out [3] = 0.499895

A.7　第 10 章实例

例 1：假设有一个（5，8）Shamir 秘密共享方案，任何一个方案都是对素数 p = 987541 求模

的。5 个共享方案是

(9853, 853), (4421, 4387), (6543, 1234), (93293, 78428), (12398, 7563),

试找到这个密秘。

解：方法一：首先，通过这 5 点找到拉格朗日插值多项式：

In [1]:= **InterpolatingPolynomial [{{9853, 853}, {4421, 4387}, {6543, 1234}, {93293, 78428}, {12398, 7563}}, x]**

Out [1]= $853 + (-\frac{1767}{2716} + (+\frac{2406987}{9538347560} + (-\frac{8464915920541}{3130587195363428640000}) - \frac{49590037201346405337547(-93293 + x)}{13378864151099487659488222679760000}(-6543 + x))(-4421 + x))(-9853 + x)$

现在计算当 $x = 0$ 时的常数项（当 $x = 0$ 使用/. x - >0 计算）：

In [2]:= **% /. x - >0**

Out [2]:= $\frac{20448432615404498323011459243394428259 1}{22298106918499146099147037799600000}$

把它转换为对 987541 取模的整数形式，我们就得到分母的乘法逆元素：

In [3]:= **PowerMod [Denominator [%], -1, 987541]**

Out [3]=509495

现在，多次乘以分子以便得到预期的整数：

In [4]:= **Mod [Numerator [%%] * %, 987541]**

Out [4]=678987

因此，678987 就是这个密秘。

方法二：在文本中建立矩阵等式，并求出多项式对 987541 求模后的系数：

In [5]:= **Solve [{{{1, 9853, 9853^2, 9853^3, 9853^4}, {1, 4421, 4421^2, 4421^3, 4421^4}, {1, 6543, 6543^2, 6543^3, 6543^4}, {1, 93293, 93293^2, 93293^3, 93293^4}, {1, 12398, 12398^2, 12398^3, 12398^4}} . {{s0}, {s1}, {s2}, {s3}, {s4}} == {{853}, {4387}, {1234}, {78428}, {7563}}, Modulus == 987541}, Mode- > Modular]**

Out [5] = {{Modulus-> 987541, s0-> 678987, s1-> 14728, s2-> 1651, s3-> 574413, s4 >456741}}

常数项为 678987，即密秘。

A.8 第 11 章实例

例 1：这是一个游戏。它本质上是 11.2 节中电话扑克游戏的简单版。有五张牌：ten, jack, queen, king, ace（译者注：即为 10, J, 小王, 大王, A）。它们已经洗过了并且通过提升它们到一个随机的指数模素数 24691313099 后来隐蔽它们。你也许能猜出哪一个是 ace。开始，选一个随机指数。在 *khide* 后面使用分号以便不被迷惑，看看 *k* 的什么值正被使用。

In [1]:= **k = khide;**

现在，改变这个被隐藏的牌的位置（它们的数字已被提升为第 *k* 次幂对 *p* 取模）：

In [2]:= **shuffle**

Out [2]= {14001090567, 16098641856, 23340023892, 20919427041, 7768690848}

这五张牌没有一张看起来像 ace；因为它们的数字已被提升为幂次方对素数取模。随便猜一个，让我们看看是否正确。

In [3]:= **reveal [%]**

Out [3]= {ten, ace, queen, jack, king}

再玩一次：

In [4]:= **k = khide;**

In [5]:= **shuffle**

Out [5]= {13015921305, 14788966861, 23855418969, 22566749952, 8361552666}

继续进行猜测（注：因为使用了不同的随机指数，所以得到的结果是不同的）。你幸运吗？

In [6]:= **reveal [%]**

Out [6]= {ten, queen, ace, king, jack}

或许你需要些帮助，再玩一次：

In [7]:= **k = khide;**

In [8]:= **shuffle**

Out [8]= {13471751030, 20108480083, 8636729758, 14735216549, 11884022059}

现在我们来寻找一些有价值的帮助：

In [9]:= **advise [%]**

Out [9]=3

由此我们知道第三张牌为 ace，再来看看（注:%%用于表示从下一个直到最后的输出）：

In [10]:= **reveal [%%]**

Out [10]= {jack, ten, ace, queen, king}

怎样做呢？读 11.2 节关于“怎样作弊”的部分。注意如果把牌的数字提升至 $(p-1)/2$ 次幂对 p 取模，可以得到：

In [11] := **PowerMod [{200514, 10010311, 1721050514, 11091407, 10305}, (24691313099 -1) /2, 24691313099]**

Out [11]= {1, 1, 1, 1, 24691313098}

因此，只有 ace 是二次方程模 p 的有效根。

A.9 第 15 章实例

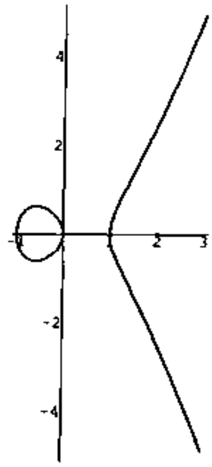
例 1：在这一章所有涉及的椭圆曲线都是对 n 取模的。然而，为了能形象地显示加法定律发生了什么，使用实数的椭圆曲线图是很有用的，即使这样的图片不存在模 n 的值。因此，先来绘制函数 $y^2 = x(x-1)(x+1)$ 的椭圆曲线。

首先，导入图形包：

In [1]:= **<<Graphics \`ImplicitPlot\`**

为了绘制曲线，我们指定 $-1 \leqslant x \leqslant 3$：

In [2]: = **ImplicitPlot [y^2 == x * (x - 1) * (x + 1), {x, -1, 3}]**



图形

例 2：把点 (1, 3) 和 (3, 5) 加到椭圆曲线 $y^2 \equiv x^3 + 24x + 13 \pmod{29}$ 上。

In [3]:= **addell [{1, 3}, {3, 5}, 24, 13, 29]**

Out [3]= {26, 1}

可以验证点 (26, 1) 在曲线 $26^3 + 24 \cdot 26 + 13 \equiv 1^2 \pmod{29}$ 上。

例 3：在上例曲线的无穷大处添加点 (1, 3)。

In [4]:= **addell [{1, 3}, {" infinity"," infinity"}, 24, 13, 29]**

Out [4]= {1, 3}

正如我们所想的，在无穷大处加点生成了点 P 且返回点 P。

例 4：设 $P = (1,3)$ 为椭圆曲线 $y^2 \equiv x^3 + 24x + 13 \pmod{29}$ 上的点，求 $7P$。

In [5]:= **multell [{1, 3}, 7, 24, 13, 29]**

Out [5]= **{15, 6}**

例 5：对于上例曲线来说，当 $k = 1,2,3,\ldots, 40$ 时，求 $k(1,3)$ 。

In [6]:= **multsell [{1, 3}, 40, 24, 13, 29]**

Out [6]= {{1, 3}, {11, 10}, {23, 28}, {0, 10}, {19, 7}, {18, 19}, {15, 6}, {20, 24}, {4, 12}, {4, 17}, {20, 5}, {15, 23}, {18, 10}, {19, 22}, {0, 19}, {23, 1}, {11, 19}, {1, 26}, {"infinity","infinity"}, {1, 3}, {11, 10}, {23, 28}, {0, 10}, {19, 7}, {18, 19}, {15, 6}, {20, 24}, {4, 12}, {4, 17}, {20, 5}, {15, 23}, {18, 10}, {19, 22}, {0, 19}, {23, 1}, {11, 19}, {1, 26}, {"infinity","infinity"}, {1, 3}, {11, 10}}

注意，每一个 19 的倍数点是怎样重复的。

例 6：前面的 4 个例子是对素数 29 取模。如果我们用一个可分解的数来计算的话，情形会变得很复杂，因为我们可能在无限地对两个因子进行计算，或者可能是在无限次地对其中一个因子取模，而不理会另一个。因此，如果后面这种情况发生的话就要停止计算，需要扩展一个因子。例如，要计算 $12P$，这里 $P = (1,3)$ 在椭圆曲线 $y^2 \equiv x^3 - 5x + 13 \pmod{209}$ 上：

In [7]:= **multell [{1, 3}, 12, −5, 13, 11 * 19]**

Out [7]= {" factor = ", 19 }

现在，来计算连续的乘积看看会发生什么：

In [8]:= **multsell [{1, 3}, 12, −5, 13, 11 * 19]**

Out [8]= {{1, 3}, {91, 27}, {118, 133}, {148, 182}, {20, 35}, {" factor = ", 19}}

当计算到 $6P$ 的时候，结束在无限大 mod 19 处。下面来看看对 209 的两个素数因子即 19 和 11 取模会发生什么：

In [9]:= **multsell [{1, 3}, 12, −5, 13, 19]**

Out [9]= {{1, 3}, {15, 8}, {4, 0}, {15, 11}, {1, 16}, {"infinity", "infinity"}, {1, 3}, {15, 8}, {4, 0}, {15, 11}, {1, 16}, { "infinity", "infinity" } }

In [10]:= **multsell [{1, 3}, 20, −5, 13, 11]**

Out [10]= {{1, 3}, {3, 5}, {8, 1}, {5, 6}, {9, 2}, {6, 10}, {2, 0}, {6, 1}, {9, 9}, {5, 5}, {8, 10}, {3, 6}, {1, 8}, {"infinity","infinity"}, {1, 3}, {3, 5}, {8, 1}, {5, 6}, {9, 2}, {6, 10}}

经历了 6 步之后，到达了无限大模 19 的位置，但是用了 14 步才到达无限大模 11 的位置。为了找到 $6P$，我们反转一个数字，用 0 对 19 取模并且非零数对 11 取模。这样做是不可能的，然而它产生了因子 19。这正是椭圆曲线因数分解法的基础。

例 7：使用椭圆曲线法因数分解 193279 。

解：首先，需要选择一些任意的椭圆曲线并在每个曲线上找一点。例如，设 $P=(1,2)$ 和椭圆曲线 $y^2 \equiv x^3 - 10x + b \pmod{193279}$。为使 P 位于曲线上，我们取 $b=28$。将有

$$y^2 \equiv x^3 + 11x - 11, P = (1,1)$$
$$y^2 \equiv x^3 + 17x - 14, P = (1,2)$$

现在我们计算点 P 的倍数，用 $p-1$ 方法来模拟，因此我们选择边界 B，令 $B=12$，计算 $B!P$。

In [11]:= **multell [{2, 4}, Factorial [12], −10, 28, 193279]**

Out [11]= {factor = , 347}

In [12]:= **multell [{1, 1}, Factorial [12], 11, −11, 193279]**

Out [12]= {13862, 35249}

In [11]:= **multell [{1, 2}, Factorial [12], 17, −14, 193279]**

Out [11]= {factor = , 557}

让我们更详细地分析在这些例子中发生了什么。

在第一个曲线上，$266P$ 结束在无穷大模 557 处，$35P$ 结束在无穷大模 347 上。因为 $266=2\cdot7\cdot19$，它是一个比 $B=12$ 大的素因子，所以 $B!P$ 不是无穷大模 557 的，但是 35 可以除尽 $B!$，因此 $B!P$ 是无穷大模 347 的。

在第二个曲线上，$356P$ = 无穷大模 347 ，$561P$ = 无穷大模 557。因为 $356=4\cdot89$ 和 $561=3\cdot11\cdot17$ ，我们不能找到用这个曲线的因数分解。

第三个曲线是令人惊奇的。有 $331P$ = 无穷大模 347 和 $272P$ = 无穷大模 557。由于 331 是素数并且 $272=16\cdot17$，我们不可能用这个曲线找到因数分解。然而，巧的是在计算 $B!P$ 时中间步骤仍产生了因数分解。下面就是所发生的。在第一步，程序要求增加点（184993,

13462）和（20678，150484）。这两个点都是和 557 同余的，但与 347 不同余，因此，倾斜穿越这二点的线被定义为模 347 的，但确是 0/0 mod 557 的。当我们试着找出分母对 193279 取模的乘积逆元素时，最大公约数算法会产生因子 557，这一现象非常少有。

例 8：这里是如何产生 15.5 节中 *ElGamal* 加密体制的椭圆曲线的例子。欲知更多的细节，参见原文。椭圆曲线是 $y^2 \equiv x^3 + 3x + 45 \pmod{8831}$，点是 $G = (4,11)$。艾丽斯的消息在点 $P_m = (5,1743)$ 上。

鲍勃已经选择秘密随机数 $a_B = 3$，而且已经计算 $a_b G$：

In [15]:= **multell [{4, 11}, 3, 3, 45, 8831]**

Out [15]= {413, 1808}

鲍勃公开这个点。艾丽斯选择随机数 $k = 8$，并计算 kG 和 $P_m + k(a_B G)$：

In [16]:= **multell [{4, 11}, 8, 3, 45, 8831]**

Out [16]= {5415, 6321}

In [17]:= **addell [{5, 1743}, multell [{413, 1808}, 8, 3, 45, 8831], 3, 45, 8831]**

Out [17]= {6626, 3576}

艾丽斯传送点（5415,6321）和（6626,3576）给鲍勃，她用 a_B 乘以第一个点：

In [18]:= **multell [{5415, 6321}, 3, 3, 45, 8831]**

Out [18]= {673, 146}

然后鲍勃减去艾丽斯发送给他的最后一个点的结果。注意，他所做的减法是通过加第二个被丢弃点来实现的：

In [19]:= **addell [{6626, 3576}, {673, −146}, 3, 45, 8831]**

Out [19]= {5, 1743}

因此鲍勃正确地得到了艾丽斯的信息。

例 9：让我们用 15.5 节 DiffieHellman 密钥交换的例子中的方法再产生一些数据：椭圆曲线是 $y^2 \equiv x^3 + x + 7206 \pmod{7211}$，点是 $G = (3,5)$。艾丽斯选择她的秘密数 $N_A = 12$，鲍勃选择他的秘密数 $N_B = 23$。艾丽斯计算

In [20]:= **multell [{3, 5}, 12, 1, 7206, 7211]**

Out [20]= {1794, 6375}

她发送点（1794，6375）给鲍勃。同时，鲍勃计算

In [21]:= **multell [{3, 5}, 23, 1, 7206, 7211]**

Out [21]= {3861, 1242}

并发送点（3861，1242）给艾丽斯。艾丽斯用 N_A 乘以她所收到的值，同时鲍勃用 N_B 乘以他所收到的值：

In [22]:= **multell [{3861, 1242}, 12, 1, 7206, 7211]**

Out [22]= {1472, 2098}

In [23]:= **multell [{1794, 6375}, 23, 1, 7206, 7211]**

Out [23]= {1472, 2098}

因此，艾丽斯和鲍勃得到了同样的密钥。

Maple 实例

本附录中的计算机实例是用 Maple 写的。如果你可以得到 Maple，建议在你的计算机上试用一下，如果得不到 Maple，仍然可以看这些例子。它们为本书所涉及到的若干概念提供了实例。为了获得 Maple 的开始信息，见 B.1 节。要获得包含 Maple 常用命令的记录集，可到如下地址下载：

http：//www.prenhall.com/washington

B.1 Maple 入门

1. 打开链接 *http：//www.prenhall.com/washington*，下载关于 Maple 的记录集文件 math.mws。

2. 打开 Maple（在 Unix 机器上，使用命令 xmaple；在其他的系统上，单击 Maple 图标），在 Maple 窗口的顶端命令栏下，使用 File 下的菜单项打开 math.mws（当你下载时，也许它自动地被执行；这取决于你的计算机设置）。

3. 在 math.mws 的前台，按下键盘上的 Enter 或 Return 键。将会装载后面的例子所必需的函数和程序包。忽视任何关于名称被再定义的警告信息。

4. 单击命令栏顶端的 File 项，移动箭头到 New 项并单击，一个新的记录区将出现在 math.mws 上方，math.mws 的所有命令仍将继续工作。

5. 如果想要给新的记录区命名，使用 File 菜单下的 Save As...，然后以 .mws 为后缀保存。

6. 现在已准备好使用 Maple 了，如果你想尝试一些简单的东西，键入 1 + 2 * 3 + 4^5；（别忘了分号）然后按下 Enter/Return 键。应该出现结果 1031（它是 $1+2\cdot 3+4^5$ 的结果）。

7. 翻到 B.3 节计算机的例子，尝试敲进一些命令，结果应该和例子中的结果相同。注意所有的命令以分号结束（或者，使用一个冒号结束输出）。按 Return 或 Enter 键使 Maple 运算表达式。

8. 如果想要删除部分记录，移动箭头到窗口左边的黑线并且双击左键。通过单击 Back Space 键或者单击命令栏顶部的 Edit 按钮后再单击出现的菜单中 Cut 按钮，高亮显示的部分就可以被删除。

9. 若要保存记录，可以单击命令栏上的 File，然后在出现的菜单中单击 Save 按钮。

10. 单击命令栏顶部的 File，在显示的菜单中单击 Print 按钮，就可以打印记录（这时将

看见在第 4 步打开一个新记录区的好处；如果你没打开，那么在 math. mws 上所有的命令都会被打印）。

11. 如果使用的命令有错误并且得到一条警告消息，可以再编辑命令并且按 Return 或 Enter 键再试一次，不需要重新键入所有的东西。

12. 通过顶端的命令栏查看可用的命令。举例来说，单击 Option，然后出现 Output Display，允许你改变输出格式。在例子中，我们使用 Standard Math Notation 选项。

13. 如果你正在寻求帮助或用一个命令完成一些事，试试在命令栏顶端的 Help 菜单。如果你能猜测一些函数的名字，还有另外的方法。举例来说，要获得关于 gcd 的信息，键入`? gcd`（没有分号），然后按 Return 或 Enter 键。

B.2 部分命令

下列是用于实例中的部分 Maple 指令，某些命令（像 `phi` 等）是 Maple 的组成部分。其他命令（像 `addell`）是在地址 *http://www.prenhall.com/washington* 处的 Maple 记录集上得来的。每一个命令都以分号结束。如果你想抑制输出，可以用冒号代替。

函数的自变量用被圆括号括住，向量用方括号括住。输入 `matrix(m,n,[a,b,c,…,z])` 给出了用第一行 a，b，... 和最后一行 ... z 表示的 $m \times n$ 矩阵。要使两矩阵 A、B 相乘，键入 `evalm (A& * B)`。

如果需涉及前面的输出，可使用%。下一个最近的输出是%%，等等。注意，% 表示最近的输出，不是最后显示的那一行。如果要经常输出，对它命名是比较好的。举例来说，`g :=phi(12345)` 定义 g 是 $\phi(12345)$ 的值。注意，当正在用这种方法分配一个值给一个变量时，应该在等号之前使用一个冒号。遗漏了冒号会引起一个很难发现的错误。

求幂是用 `a^b` 表示。然而，我们需要对非常大的指数使用指数求模来处理。在这种情况下，使用 `a&^b mod n`。

下列部分命令要求由某些 Maple 程序包导入

```
with(numtheory),with(linalg),with(plots),with(combinat)
```

当装载 math. mws 记录集时这些命令也被装载上了。然而，如果想在没有装载记录集的环境下使用一些命令，如 `nextprime`，首先，要键入 `with (numtheory):` 来装载程序包（只需一次），然后就可以使用一些函数例如 `nextprime`，`isprime`，等等。如果键入的是 `with (numtheory);` 使用的是分号，将会得到一个关于包中函数的清单。

下面是用于实例中的部分命令。为了便于参考，将它们列在这里。具体的使用参见例子。我们用 txt 表示一串字母，这些字母串用引号括起（" string"）。

`addell([x,y],[u,v],a,b,n)` 为求椭圆曲线 $y^2 \equiv x^3 + ax + b \pmod n$ 上的点 (x,y) 和点 (u,v) 之和。整数 n 应为奇数。

`affinecrypt (txt, m, n)` 利用 $mx + n$ 仿射加密 txt。

`allshifts (txt)` 给出 txt 的所有 26 个移位转换。

`chrem([a,b,...],[m,n,...])` 给出了同时满足 $x \equiv a \pmod m, x \equiv b \pmod n$，... 的解。

choose(txt,m,n) 列出与 $n(\bmod m)$ 同余的 txt 的特征。

coinc(txt,n) 是 txt 和 txt 移动 n 位后相匹配的数字。

corr(v) 表示向量 v 与字母频率向量移位 26 次后产生的点积。

phi(n) 计算 $\phi(n)$（n 值不能取太大）。

igcdex(m,n,'x','y') 计算满足条件 $mx + ny = \gcd$ 的 m 和 n 的 gcd。为了获得 x 和 y，在这一行或随后的行中键入 x; y;。

ifactor(n) 因数分解 n。

frequency(txt) 列出 a 到 z 在 txt 中出现的次数。

gcd(m,n) 是求 m 和 n 的最大公约数。

inverse(M) 求矩阵 M 的逆矩阵。

lfsr(c,k,n) 给出了由向量 c 表示系数的递归所产生的 n 位序列，位数的初始值由向量 k 给出。

lfsrlength(v,n) 测试向量 v 的位数，看它是否由长度最多为 n 的递归产生。

lfsrsolve(v,n) 给定一个产生二进制向量 v 的递归长度 n，计算这个递归的系数。

max(v) 是列表 v 的最大元素。

a mod n 是 $a(\bmod n)$ 的值。

multell([x,y],m,a,b,n) 计算椭圆曲线 $y^2 \equiv x^3 + ax + b(\bmod n)$ 上的点 (x,y) 的 m 次方。

multsell([x,y],m,a,b,n) 计算椭圆曲线 $y^2 \equiv x^3 + ax + b(\bmod n)$ 上的点 (x,y) 第一个 m 的倍数。

nextprime(x) 给出下一个大于 x 的素数。

num2text(n) 将数字 n 变成字母，每一对连续的数字最大为 26，空为 00，a 是 01，z 是 26。

primroot(p) 指出素数 p 的一个本原根。

shift(txt,n) txt 移动 n 位。

text2num(txt) 用 space = 00, a = 01, ..., z = 26 的形式将 txt 变换成数字。

vigenere(txt,v) 使用向量 v 给出 txt 的 Vigenère 密码。

vigvec(txt,m,n) 给出文本中字母 a 到 z 与 $n(\bmod m)$ 同余时出现的频率。

B.3　第 2 章实例

例 1：使用移位密码获得密文 kddkmu，尝试所有的可能性来解密它。

```
> allshifts ("kddkmu");

                    "kddkmu"
                    "leelnv"
                    "mffmow"
                    "nggnpx"
                    "ohhoqy"
```

```
"piiprz"
"qjjqsa"
"rkkrtb"
"sllsuc"
"tmmtvd"
"unnuwe"
"voovxf"
"wppwyg"
"xqqxzh"
"yrryai"
"zsszbj"
"attack"
"buubdl"
"cvvcem"
"dwwdfn"
"exxego"
"fyyfhp"
"gzzgiq"
"haahjr"
"ibbiks"
"jccjlt"
```

正如你所看到的，attack 是出现在列表上的惟一一个单词，这就是明文。

例 2：用仿射函数 $7x+8$ 对明文 cleopatra 加密：

```
>affinecrypt ("cleopatra", 7, 8);

                    "whkcjilxi"
```

例 3：密文 mzdvezc 是用仿射函数 $5x+12$ 加密的，试对它解密。

解：首先，解 $y \equiv 5x+12 \pmod{26}$，得 $x \equiv 5^{-1}(y-12)$。我们需要找到 $5 \pmod{26}$ 的逆元素：

```
>5&^(-1) mod 26;

                    21
```

因此，$x \equiv 21(y-12) \equiv 21y-12\cdot 21$。把 $-12\cdot 21$ 转换为标准形式：

```
> -12*21 mod 26;

                    8
```

因此，解密函数是 $x \equiv 21y+8$，解密信息为：

```
>affinecrypt ("mzdvezc", 21, 8);

                    "anthony"
```

要是你觉得迷惑，明文是用如下形式加密的：

```
>affinecrypt ("anthony", 5, 12);

                    "mzdvezc"
```

例 4：这是一个关于文本的 Vigenère 密码的例子，让我们看看怎么用 2.3 节中的方法把它解密成数据，为了方便，我们已经存储了名为 vvhq 的加密文本。

```
> vvhq;
  vvhqwvvrhmusgjgthkihtssejchlsfcbgvwcrlryqtfsvgahwkcuhwauqlqhnslrljshbltspi
  sprdxljsveeghlqwkasskuwepwqtwvspgoelkcqyfnsvwljsniqkgnrgybwlwgoviokhkazkqk
  xzgyhcecmeiujoqkwfwvefqhkijrclrlkbienqfrjljsdhgrhlsfqtwlauqrhwdmwlgusgikkf
  lryvcwvspgpmlkassjvoqxeggveyggzmljcxxljsvpaivwikvrdrygfrjljslveggveyggeiap
  uuisfpbtgnwwmuczrvtwglrwugumnczvile
```

找出密文中字母的频率：

```
> frequency (vvhq);
[8, 5, 12, 4, 15, 10, 27, 16, 13, 14, 17, 25, 7, 7, 5, 9, 14, 17, 24, 8, 12, 22,
22, 5, 8, 5]
```

我们来计算移位 1，2，3，4，5，6 个位置的相同值：

```
> coinc (vvhq, 1);
                                          14
> coinc (vvhq, 2);
                                          14
> coinc (vvhq, 3);
                                          16
> coinc (vvhq, 4);
                                          14
> coinc (vvhq, 5);
                                          24
> coinc (vvhq, 6);
                                          12
```

我们得出密钥长度可能是 5 位的结论，下面来看一看第 1 个，第 6 个，第 11 个，等等字母（即，这些位置的字母都是和 1 mod 5 同余的）：

```
> choose (vvhq, 5, 1);
"vvuttcccqgcunjtpjgkuqpknjkygkkgcjfqrkqjrqudukvpkvggjjivgjggpfncwuce"
> frequency (% );
[0, 0, 7, 1, 1, 2, 9, 0, 1, 8, 8, 0, 0, 3, 0, 4, 5, 2, 0, 3, 6, 5, 1, 0, 1, 0]
```

将其表示为一个频率向量为：

```
> vigvec (vvhq, 5, 1);
[0., 0., .1044776119, .01492537313, .01492537313, .02985074627, .1343283582, 0.,
.01492537313, .1194029851, .1194029851, 0., 0., .04477611940, 0., .05970149254,
.07462686567, .02985074627, 0., .04477611940, .08955223881, .07462686567,
.01492537313, 0., .01492537313, 0。]
```

这个向量与字母频率向量移位后的点积计算如下：

```
> corr (% );
  .02501492539, .03910447762, .07132835821, .03882089552, .02749253732,
  .03801492538, .05120895523, .03014925374, .03247761194, .04302985074,
  .03377611940, .02985074628, .03426865672, .04456716420, .03555223882,
  .04022388058, .04343283582, .05017910450, .03917910447, .02958208957,
  .03262686569, .03917910448, .03655223881, .03161194031, .04883582088,
```

```
.03494029848。
```

第三项是最大值，但有时最大的位置点是很难确定的。一种定位的方法是：

```
> max (% );
                                .07132835821
```

现在很容易从清单中看到并找出这个数（它通常只出现一次）。由于它在第三个位置出现，这个 Vigenère 密码是由首字母移动了 2 位形成的，相应的字母为 c。这个程序和刚刚使用过的非常相似（使用 vigvec(vvhq,5,2),...,vigvec(vvhq,5,5)），这表明其他的移位很可能是 14，3，4，18。我们用正确的密钥来验证一下其正确性。

```
> vigenere (vvhq, - [2, 14, 3, 4, 18]);
```

```
themethodusedforthepreparationandreadingofcodemessagesissimpleintheextreme
andatthesametimeimpossibleoftranslationunlessthekeyisknowntheeasewithwhich
thekeymaybechangedisanotherpointinfavoroftheadoptionofthiscodebythosedesir
ingtotransmitimportantmessageswithouttheslightestdangeroftheirmessagesbein
greadbypoliticalorbusinessrivalsetc
```

回顾一下，明文最初是用如下命令加密的

```
> vigenere (% , [2, 14, 3, 4, 18]);
```

```
vvhqwvvrhmusgjgthkihtssejchlsfcbgvwcrlryqtfsvgahwkcuhwauglqhnslrljshbltspi
sprdxljsveeghlqwkasskuwepwqtwvspgoelkcqyfnsvwljsniqkgnrgybwlwgoviokhkazkqk
xzgyhcecmeiujoqkwfwvefqhkijrclrlkbienqfrjljsdhgrhlsfqtwlauqrhwdmwlgusgikkf
lryvcwvspgpmlkassjvoqxeggveyggzmljcxxljsvpaivwikvrdrygfrjljslveggveyggeiap
uuisfpbtgnwwmuczrvtwglrwugumnczvile
```

例 5：密文

22,09,00,12,03,01,10,03,04,08,01,17

是通过矩阵

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 10 \end{pmatrix}$$

使用希尔密码加密的，试对它解密。

解：输入一个矩阵有几种方法。一种方法如下：可以将一个 2 × 2 的矩阵$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$作为 matrix(2,2,[a,b,c,d]) 输入。键入 evalm(M&*N) 表示矩阵 M 和矩阵 N 相乘。evalm(v&*M) 表示向量 v 右乘矩阵 M。

这儿有加密矩阵。

```
>M:=matrix (3, 3, [1, 2, 3, 4, 5, 6, 7, 8, 10]);
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 10 \end{bmatrix}$$

我们需要使矩阵对 26 取模，然后转置：

```
>invM:=map (x - >x mod 26, inverse (M));
```

$$\begin{bmatrix} 8 & 16 & 1 \\ 8 & 21 & 24 \\ 1 & 24 & 1 \end{bmatrix}$$

命令 map(x -> x mod 26,E) 取出表达式 E 中的每一个数字并减去它模 26 的结果。下面是矩阵模 26 后的转置。可验证如下：

```
> M& * invM;
```

$$\begin{bmatrix} 27 & 130 & 52 \\ 78 & 313 & 130 \\ 130 & 520 & 209 \end{bmatrix}$$

```
> map (x - >x mod 26,% );
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

为了解码，我们把密文分成 3 个字母一组，且每一组右乘刚才计算出来的转置矩阵：

```
> map (x - >x mod 26, evalm ( [22, 09, 00] &* invM));
                                   [14, 21, 4]
> map (x - >x mod 26, evalm ( [12, 03, 01] &* invM26));
                                   [17, 19, 7]
> map (x - >x mod 26, evalm ( [10, 03, 04] &* invM26));
                                   [4, 7, 8]
> map (x - >x mod 26, evalm ( [08, 01, 17] &* invM26));
                                   [11, 11, 23]
```

因此，明文是 14,21,4,17,19,7,4,7,8,11,11,23。把它变回字母，可以得到 overthehillx。注意，在明文最后加的那个 x 是为了凑成 3 个字母的字串。

例 6：计算递归式子 $x_{n+5} \equiv x_n + x_{n+2} \pmod 2$ 的前 50 项，初始值为 0,1,0,0,0。

解：系数的向量是 [1,0,1,0,0]，初始向量是[0,1,0,0,0]。键入

```
> lfsr ( [1, 0, 1, 0, 0], [0, 1, 0, 0, 0], 50);
[0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1,
1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1]
```

例 7：假设 LFSR 序列的前 20 项是 1,0,1,0,1,1,1,0,0,0,0,1,1,1,0,1,0,1,0,1。使用递归算法生成这个序列。

解：首先，我们确定递归运算的长度。命令 lfsrlength[v,n] 用于计算出现在 2.11 节中的前 n 个阵列的模 2 行列式。

```
> lfsrlength ( [1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1], 10);
                                   [1, 1]
                                   [2, 1]
                                   [3, 0]
                                   [4, 1]
                                   [5, 0]
                                   [6, 1]
                                   [7, 0]
                                   [8, 0]
                                   [9, 0]
                                   [10, 0]
```

最后的一个非零行列式是第六项，所以我们猜测递归的长度为 6，可以确定系数为：

```
> lfsrsolve ( [1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1], 6);
                    [1, 0, 1, 1, 1, 0]
```

得到递归式如下

$$x_{n+6} \equiv x_n + x_{n+2} + x_{n+3} + x_{n+4} \pmod 2)。$$

例 8：密文 0,1,1,0,1,0,1,0,1,0,0,1,1,0,0,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,0,1,0,0,0,1,0,1,1,0 是由明文对 2 取模再加上 LFSR 的输出得到的（如，用明文同 LFSR 的输出进行异或）。假设知道明文是以 1,1,1,1,1,1,0,0,0,0,0,0,1,1,1,0,0 开始的，找出余下的明文。

解：用已知的部分明文异或密文可以获得 LFSR 输出的开始部分：

```
> [1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0]
   + [0 , 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1] mod 2;
      [1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1]
```

这是 LFSR 输出的开始。现在来求递归的长度。

```
> lfsrlength (% , 8);
                    [1, 1]
                    [2, 0]
                    [3, 1]
                    [4, 0]
                    [5, 1]
                    [6, 0]
                    [7, 0]
                    [8, 0]
```

我们猜测长度是 5。为找出递归的系数：

```
> lfsrsolve (% % , 5);
                    [1, 1, 0, 0, 1]
```

现在我们使用刚刚得出的系数再加上 LFSR 输出的最初五项可以得到 LFSR 的完全输出：

```
> lfsr ( [1, 1, 0, 0, 1], [1, 0, 0, 1, 0], 40);
[1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1,
0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0]
```

当用密文与 LFSR 输出进行异或，就可以得到明文：

```
> % + [0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0] mod 2;
[1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1,
1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0]
```

这就是明文。

B.4 第 3 章实例

例 1：找出 gcd(23456,987654)。

```
> gcd (23456, 987654);
                    2
```

例 2：在整数范围内求 x，y，使之满足 $23456x+987654y=2$。

```
igcdex (23456,987654,'x','y');
```

2

```
> x; y;
```

−3158

75

这就意味着 2 是最大公约数并且 $23456\cdot(-3158)+987654\cdot 75=2$（命令 igcdex 是求整数 gcd 的扩展，Maple 也可以计算 gcd 多项式）。除 'x','y' 之外的变量名也可以使用，如果这些字母要用在其他地方，如在多项式中。我们能依下面各项清除 x 的值：

```
> X:='X';
```

X:= X

例 3：计算 $234\cdot 456 \pmod{789}$。

```
> 234 * 456 mod 789;
```

189

例 4：计算 $234567^{876543} \pmod{565656565}$。

```
> 234567&^876543 mod 565656565;
```

473011223

例 5：求 $87878787 \pmod{9191919191}$ 的乘法逆元素。

```
> 87878787&^(-1) mod 9191919191;
```

7079995354

例 6：求解 $7654x\equiv 2389 \pmod{65537}$。

解：方法一：

```
> solve (7654 * x =2389, x) mod 65537;
```

43626

方法二：

```
> 2389/7654 mod 65537;
```

43626

例 7：找出满足下列条件的 x：

$$x\equiv 2 \pmod{78},x\equiv 5 \pmod{97},x\equiv 1 \pmod{119}。$$

```
> chrem ( [2, 5, 1], [78, 97, 119]);
```

647480

检验结果：

```
> 647480 mod 78; 647480 mod 97; 647480 mod 119;
```

2

5

1

例 8：把 123450 因数分解为素数。

```
> ifactor (123450);
```

$(2)(3)(5)^2(823)$

即 $123450=2^1 3^1 5^2 823^1$。

例 9：求 $\phi(12345)$ 的值。

```
> phi (12345);
```

$$6576$$

例 10：找出素数 65537 的本原根。

```
> primroot (65537);
```

$$3$$

因此，3 就是素数 65537 的本原根。

例 11：找出矩阵 $\begin{pmatrix} 13 & 12 & 35 \\ 41 & 53 & 62 \\ 71 & 68 & 10 \end{pmatrix}$ (mod 999) 的逆矩阵。

解：首先转换不带模的矩阵，然后减去矩阵模 999 的余数：

```
> inverse (matrix (3, 3, [13, 12, 35, 41, 53, 62, 71, 68, 10]));
```

$$\begin{bmatrix} \frac{3686}{34139} & -\frac{2260}{34139} & \frac{1111}{34139} \\ -\frac{3992}{34139} & \frac{2355}{34139} & -\frac{629}{34139} \\ \frac{975}{34139} & \frac{32}{34139} & -\frac{197}{34139} \end{bmatrix}$$

```
> map (x - >x mod 999,% );
```

$$\begin{bmatrix} 772 & 472 & 965 \\ 641 & 516 & 851 \\ 150 & 133 & 149 \end{bmatrix}$$

这是矩阵模 999 的逆。

例 12：求 26951623672 模素数 p = 98573007539 的一个平方根.

解：由于 $p \equiv 3 (\text{mod } 4)$，我们可以使用 3.9 节中的方法：

```
> 26951623672&^((98573007539 +1)/4)mod 98573007539;
```

$$98338017685$$

指数上额外的括号是必需的；否则，指数将被认为是 98573007539 + 1，所得的结果再除以 4。另外一个平方根是它的负数：

```
>  -% mod 98573007539;
```

$$234989854$$

例 13：令 n = 34222273 = 9803 · 3491，求 $x^2 \equiv 19101358 (\text{mod } 34222273)$ 的所有 4 个解。

解：首先，找一个平方根分别对两个素数因子取模，两者都与 3(mod 4) 同余：

```
> 19101358&^((9803 +1)/4)mod 9803;
```

$$3998$$

```
> 19101358&^((3491 +1)/4)mod 3491 ;
```

$$1318$$

因此，这些平方根与 ± 3998(mod 9803) 和 ± 1318(mod 3491) 是同余的。利用中国剩余定理，有 4 种方法来组合这些：

```
> chrem ( [3998, 1318], [9803, 3491]);
```

$$43210$$

```
> chrem ( [ -3998, 1318], [9803, 3491]);
```

$$8397173$$

```
> chrem ( [3998, -1318], [9803, 3491]);
                                        25825100
> chrem ( [-3998, -1318], [9803, 3491]);
                                        34179063
```

这些就是要求的 4 个平方根。

B.5 第6章实例

例 1：假如需要从 50 个数中得到一个最大的随机素数，这里有一种方法。函数 nextprime 可以寻找比 x 大的下一个素数。函数 rand(a..b)( ) 给出 a 和 b 之间的一个随机整数。结合这些，我们能找到一个素数：

```
> nextprime (rand (10^49..10^50]] ( ));
                    73050570031667109175215303340488313456708913284291
```

如果我们重复这个过程，将得到另外的素数：

```
> nextprime (rand (10^49..10^50]] ( ));
                    97476407694931303255724326040586144145341054568331
```

例 2：假如想把文本 *hellohowareyou* 转换成数字：

```
> text2num ("hellohowareyou");
                              805121215081523011805251521
```

注意这里，我们让 $a=1, b=2, \ldots, z=26$，否则 a 在信息首部时将消失。（一个更高效的方法建立在基数 27 之上，于是信息的数字形式为 $8+5\cdot 27+12\cdot 27^2+\cdots+21\cdot 27^{13}=87495221502384554951$。注：这种方法使用了更少的位数。）

假如想把它转换回字母：

```
> num2text (805121215081523011805251521);
                                     "hellohowareyou"
```

例 3：在 $n=823091, e=17$ 的条件下用 RSA 法加密信息 hi。

解：首先，将信息转换成数字：

```
> text2num ("hi");
                                        809
```

现在，把它提升到 e 次幂对 n 取模：

```
> % &^17 mod 823091;
                                       596912
```

例 4：对上例的密文解密。

解：首先，我们需要得到解密指数 d。为此，要找到 $\phi(823091)$。一种方法是：

```
> phi (823091);
                                       821184
```

另一种方法是把 n 分解成 $p\cdot q$ 的形式，然后计算 $(p-1)(q-1)$：

```
> ifactor (823091);
                                   (659) (1249)
> 658 * 1248
```

821184

由于 $de \equiv 1(\bmod \phi(n))$，有如下计算（注意，我们寻找 $e \bmod \phi(n)$ 的逆，而不是 mod n 的逆）：

```
> 17&^(-1) mod 821184;
```

48305

因此，$d = 48305$。为了解密，提升密文到 d 次幂对 n 取模：

```
> 596912&^48305 mod 823091;
```

809

最后，转换成字母：

```
> num2text (809);
```

"hi"

例 5：当 $n = 823091$ 和 $e = 17$ 时，用 RSA 对信息 hellohowareyou 加密。

解：首先，将明文转换成数字：

```
> text2num ("hellohowareyou");
```

80512121508152301180525 1521

假设我们只简单地提升到 e 次幂对 n 取模：

```
> % &^17 mod 823091);
```

447613

如果解密（从例 4 可以知道 d），可得

```
> % &^48305 mod 823091;
```

628883

这不是原来的明文。原因是明文比 n 大，因此我们已经得到了模 n 后的结果明文：

```
> 805121215081523011805251521 mod 823091;
```

628883

我们需要把明文分组，每组小于 n，在本例中，我们一次使用 3 个字母：

80512 121508 152301 180525 1521

```
> 80512&^17 mod 823091;
```

757396

```
> 121508&^17 mod 823091;
```

164513

```
> 152301&^17 mod 823091;
```

121217

```
> 180525&^17 mod 823091;
```

594220

```
> 1521&^17 mod 823091;
```

442163

密文为 757396164513121217594220442163。注意：这个密文无法还原回字母。实际上，它不与任何字母文本相对应。

分别解密每一组：

```
> 757396&^48305 mod 823091;
```

80512

```
> 164513&^48305 mod 823091;
                              121508
```

等等。

下面我们举一些大数字的例子，即在 6.5 节中讨论的关于 RSA 挑战中所涉及的例子。它们存储在名为 *rsan*，*rsae*，*rsap*，*rsaq* 的文件下：

```
> rsan;
1143816257578888676692357799761466120102182967212423625625618429357069352457
33897830597123563958705058989075147599290026879543541
> rsae;
                              9007
```

例 6：使用 *rsan* 和 *rsae* 分别对 b，ba，bar，bard 加密。

```
> text2num ("b") &^rsae mod rsan;
7094675846761266859837016499155078618287633106068523541056470411448678226171
6497200122155332348462014053287987580899263765142534
> text2num ("ba") &^rsae mod rsan;
3504513060897510032501170944987195427378820475394859306031369769822762175980
6027962270538031565564773352033671782261305796158951
> text2num ("bar") &^rsae mod rsan;
4481451286385510107600453085949210934242953160660740907036054340800084364598
6880405953102818312822586362580298784441151922606424
> text2num ("bard") &^rsae mod rsan;
2423807778511166642320286251209031739348521295905627078313499161425605432329
7179804928958073445752663026449873986877989329909498
```

观察到这些密文的长度是相同的，这使得我们很难有一种简单的方法来确定相应明文的长度。

例 7：利用因数分解 *rsan* = *rsap* · *rsaq*，求对于 RSA 挑战的解密指数，并解密密文（见 6.5 节）。

首先找到解密指数：

```
>rsad:=rsae&^(-1) mod ((rsap-1) * (rsaq-1)):
```

注意，我们在最后使用冒号是为了避免打印出它的值。如果你想看 *rsad* 的值，参见 6.5 节或不使用分号。要对以文件名为 *rsaci* 存储的密文解密，转换成字母即可：

```
> num2text (rsaci&^rsad mod rsan);
                "the magic words are squeamish ossifrage"
```

例 8：使用 *rsan* 和 *rsae* 对信息 rsaencryptsmessageswell 加密。

```
> text2num ("rsaencryptsmessageswell") &^rsae mod rsan;
9463942034900225931630582353924949641464096993400170972140435241827195065425
4365584906013966328817753539283112653197553130781884
```

例 9：对上一题的密文解码。

解：幸好我们知道 *rsad* 的解密指数，因此，我们计算

```
> % &^rsad mod rsan;
                 1819010514031825162019130519190107051923051212
```

```
> num2text (% );
                                   "rsaencryptsmessageswell"
```

假设我们在传输的时候丢失了密文的最后4位，试着对剩下的密文解密（减去4并除以10就可以将后4位移去）：

```
> (% % %   4) /10) &^rsad mod rsan;
4795299917319598866490235262952548640911363389437562984685490797058841230037
3487969657794254117158956921267912628461494475682806
```

如果试着把它转换为字母，会得到一长串的错误信息；通常明文中的一个小错误就能完全地改变解密信息并产生毫无意义的数据。

例10：已知 $n = 1131377127559031 2567$ 是两个素数的乘积，$\phi(n) = 11313771187608744400$。因数分解 n。

解：我们知道（见6.1节）p 和 q 是方程 $X^2 - (n - \phi(n) + 1)X + n$ 的两个根。因此，我们计算

```
> solve (x^2 -
(11313771275590312567 -11313771187608744400 +1) *x +11313771275590312567, x);
                                   87852787151, 128781017
```

因此，$n = 128781017 \cdot 87852787151$。我们也可以用二次配方法来求它的根。

例11：设已知 *rsae* 和 *rsad*，使用这些对 *rsan* 因数分解。

解：使用6.4节中的通用指数因数分解方法。首先，记 $rsae \cdot rsad - 1 = 2^s m$，$m$ 是奇数。一种方法是先计算 $rsae \cdot rsad - 1$，然后使它除以2直到得到一个奇数：

```
> rsae * rsad -1;
9610344196177822661569190233595838341098541290518783302506446040411559855750
8735265915617489855734299513159468043108692124583009766 4
> % /2;
4805172098088911330784595116797919170549270645259391651253223020205779927875
4367632957808744927867149756579734021554346062291504883 2
> % /2;
2402586049044455665392297558398959585274635322629695825626611510102889963937
7183816478904372463933574878289867010777173031145752441 6
```

一直这样计算六步直到得到

```
3754040701631961977175464934998374351991617691608899727541580484535765568652
6849713248288081974896210747327917204339332861165238 19
```

这个数是 m。现在选择一个随机整数 a。假设选到的是13，利用通用指数因数分解法，计算：

```
> 13&^% mod rsan;
2757436850700653059224349486884716119842309570730780569056983964703018310983
986237080052933809298479549019264358796085987055123 9
```

由于它不等于 $\pm 1 (\bmod\ rsan)$，继续对它平方直到得到 ± 1：

```
> % &^2 mod rsan;
4831896032192851558013847641872303455410409906994084622549470277665499641258
29556360352661561086864311942985740758540375122772 92
```

```
> % &^2 mod rsan;
78172814154877356579141928058754000021948787056483820917930625115215181839742056013275521913487560944732073516487722273875579363
> % &^2 mod rsan;
42836191202508728742199299040582900202976222916017767167551870216509444518239462186379470569442055101392992293082259601738228702
> % &^2 mod rsan;
                                   1
```

由于 1 以前的最后一位数不是 ±1 (mod *rsan*)，我们有一个当 $x^2 \equiv 1$ 时 $x \not\equiv \pm 1 \pmod{rsan}$ 的例子。因此，gcd($x-1$,*rsan*) 是 *rsan* 的一个非平凡因子：

```
> gcd (%% -1, rsan);
327691329932667095499619881908344614131776429679929425397982 88533
```

这就是 *rsaq*。另一个因子通过计算 *rsan/rsaq* 可获得：

```
rsan/% ;
3490529510847650949147849619903898133417764638493387843990820577
```

即 *rsap*。

例 12：设已知 $150883475569451^2 \equiv 16887570532858^2 \pmod{205611444308117}$，因数分解 205611444308117。

解：我们使用 6.3 节中的基本原理：

```
> gcd (150883475569451 -16887570532858, 205611444308117);
                                   23495881
```

这给出了一个因子。另一个因子是

```
> 205611444308117/% ;
                                   8750957
```

我们能够检验这些因子是素数，所以不能再进一步因数分解了：

```
> isprime (%%);
                                   true
> isprime (%%);
                                   true
```

例 13：用 $p-1$ 的方法因数分解 $n = 376875575426394855599989992897873239$。

解：选择 $B = 100$ 为界，令 $a = 2$，于是可以计算出 $2^{100!} \pmod n$：

```
> 2&^factorial (100)
mod 376875575426394855599989992897873239;
                    369676678301956331939422106251199512
```

然后我们计算 $2^{100!}-1$ 和 n 的 gcd：

```
> gcd (% -1, 376875575426394855599989992897873239);
                    430553161739796481
```

这就是因子 p，另一个因子 q 为：

```
> 376875575426394855599989992897873239/% ;
                    875328783798732119
```

让我们看看为什么这样计算。$p-1$ 和 $q-1$ 的分解因子是

```
> ifactor (430553161739796481 -1);
```

$$(2)^{18}(3)^7(5)(7)^4(11)^3(47)$$

```
> ifactor (875328783798732119 -1);
```

$$(2)(61)(8967967)(20357)(39301)$$

可见 100! 是 $p-1$ 的倍数，因此 $2^{100!}\equiv 1(\bmod p)$ 。然而，100! 不是 $q-1$ 的倍数，所以它有可能满足 $2^{100!}\not\equiv 1(\bmod q)$ 。因此，$2^{100!}-1$ 和 pq 都以 p 作为因子，但是只有 pq 以 q 作为因子。最大公约数 gcd 是 p。

B.6 第 8 章实例

例 1：设一间屋子里有 23 个人，至少有两个人生日相同的概率是多少？

解：没有两个人相同的概率是 $\prod_{i=1}^{22}(1-i/365)$（注意乘积是到 $i=22$ 而不是 $i=23$）。用 1 减去这个概率就是两个人生日相同的概率：

```
> 1 -mult (1. -i/365, i =1..22);
```

.5072972344

注：在求积时用 1. 代替没有小数点的 1。如果去掉这个小数点，将是计算有理数的积（试试看，就知道了）。

例 2：设一个懒惰的电话公司职员用随机的 7 位数来分配电话号码。在一个有 10000 部电话的城市，两个人分到同一号码的概率是多少？

```
> 1 -mult (1. -i/10^7, i =1..9999);
```

.9932699135

注：电话号码大约是可能电话数量的平方根的 3 倍。这就意味着概率非常高。由 8.4 节，我们推测如果有 $\sqrt{2(\ln 2)10^7}\approx 3723$ 个左右的电话，就有 50% 的号码相同的机会。下面来验证一下它的正确性：

```
> 1 -mult (1. -i/10^7, i =1..3722);
```

.4998945410

B.7 第 10 章实例

例 1：假设有一个 (5,8) Shamir 秘密共享方案。任何一个方案都是对素数 $p=987541$ 求模的。5 个共享方案是

(9853, 853), (4421, 4387), (6543, 1234), (93293, 78428), (12398, 7563),

试找到这个密秘。

解：方法一：首先，通过这 5 点找到拉格朗日插值多项式：

```
> interp ( [9853, 4421, 6543, 93293, 12398],
[853, 4387, 1234, 78428, 7563], x);
```

$$-\frac{4959003720134640533754 7}{1337886415109948769488222679760000 0}x^4$$

$$+\frac{3531308571699255777907330 7}{8919242767399658439658815119840000}x^3$$

$$-\frac{88296289783211397710768373614 81}{19112663072999268084983175256800000}x^2$$

$$+\frac{97490492304744507169508035198110 81}{44596213836998292198294075599200000}x$$

$$+\frac{2044843261540449832301145924339442825 91}{22298106918499146099147037799600000}$$

现计算当 $x=0$ 时的常数项：

```
> eval (% , x=0);
```

$$\frac{204484326154044983230114592433944282591}{22298106918499146099147037799600000}$$

需要转换为对 987541 取模的整数形式：

```
> % mod 987541;
```

678987

因此，678987 是密秘。

下面是另一种方法。在文本中建立矩阵等式并求出多项式对 987541 取模后的系数：

```
> map (x - >x mod 987541, evalm (inverse (matrix (5, 5,
[1, 9853, 9853^2, 9853^3, 9853^4,
1, 4421, 4421^2, 4421^3, 4421^4,
1, 6543, 6543^2, 6543^3, 6543^4,
1, 93293, 93293^2, 93293^3, 93293^4,
1, 12398, 12398^2, 12398^3, 12398^4]))
&* matrix (5, 1, [853, 4387, 1234, 78428, 7563])));
```

$$\begin{bmatrix} 678987 \\ 14728 \\ 1651 \\ 574413 \\ 456741 \end{bmatrix}$$

常数项是 678987，即密秘。

B.8 第 11 章实例

例 1：这是一个游戏。它本质上是 11.2 节中电话纸牌游戏的简单版。有五张牌：ten，jack，queen，king，ace（译者注：即为 10，J，小王，大王，A）。它们已经洗过了并且通过提升它们到一个随机的指数模素数 24691313099 后来隐蔽它们。你也许能猜出哪一个是 ace。开始，选一个随机指数。在 khide ( ) 后面使用冒号以便不被迷惑，看看 k 的什么值正被使用。

```
> k:=khide ():
```

现在，改变这个被隐藏的牌的位置（它们的数字已被提升为第 k 次幂对 p 取模）：

```
> shuffle (k);
```

$$[14001090567,\ 16098641856,\ 23340023892,\ 20919427041,\ 7768690848]$$

这五张牌没有一张看起来像 ace；因为它们的数字已被提升为幂次方对素数取模。随便猜一个，让我们看看是否正确。

```
> reveal (% );
```

["ten","ace","queen","jack","king"]

让我们再玩一次：

```
> k=khide ();
> shuffle (k);
```

$$[13015921305,\ 14788966861,\ 23855418969,\ 22566749952,\ 8361552666]$$

继续进行猜测（注意，因为用了不同的随机指数，所以得到的数字也不同）。你是幸运的吗？

```
> reveal (% );
```

["ten","queen","ace","king","jack"]

也许你需要一些帮助。让我们再玩一次：

```
> k=khide: ()
> shuffle (k);
```

$$[13471751030,\ 20108480083,\ 8636729758,\ 14735216549,\ 11884022059]$$

现在我们要求一些提示：

```
> advise (% );
```

$$3$$

我们被提示第 3 张牌是 ace。下面来看看（%%是用来指出下一个输出）：

```
> reveal (% % );
```

["jack","ten","ace","queen","king"]

这是如何工作的呢？见 11.2 节中“怎样作弊”的部分。注意，如果把牌的数字提升至 $(p-1)/2$ 次幂对 p 取模，可以得到：

```
> map (x - >x&^( (24691313099 -1) /2) mod 24691313099,
[200514, 10010311, 1721050514, 11091407, 10305]);
```

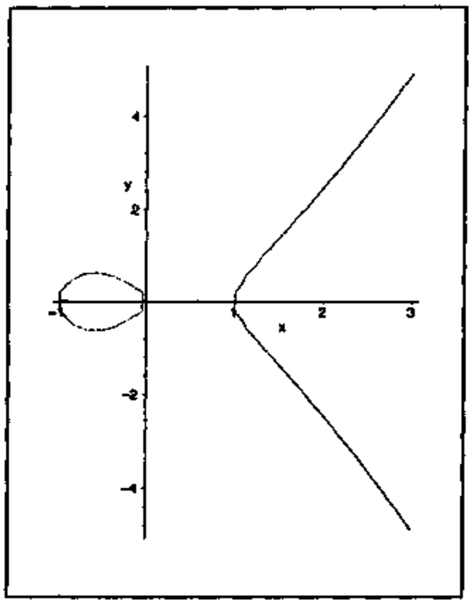
$$[1,\ 1,\ 1,\ 1,\ 24691313098]$$

因此，只有 ace 是模 p 的二次非剩余。

B.9 第 15 章实例

例 1：这一章中所有涉及的椭圆曲线都是对 n 取模的。然而，为了能形象地显示加法定律发生了什么，使用实数的椭圆曲线图是很有用的，即使这样的图片不存在模 n 的值。因此，先来绘制函数 $y^2=x(x-1)(x+1)$ 的椭圆曲线。我们指定 $-1\leqslant x\leqslant 3$，$-5\leqslant y\leqslant 5$。

```
> implicitplot (y^2 =x* (x-1) * (x+1), x= -1..3, y= -5..5);
```



例 2：把点 (1,3) 和(3,5) 加到椭圆曲线上 $y^2 \equiv x^3 + 24x + 13 \pmod{29}$ 上。

```
> addell ( [1, 3], [3, 5], 24, 13, 29);
                                        [26, 1]
```

可以验证点 (26,1) 在曲线 $26^3 + 24 \cdot 26 + 13 \equiv 1^2 \pmod{29}$ 上。

例 3：在上例曲线上的无限大处添加点 (1, 3)。

```
> addell ( [1, 3], ["infinity","infinity"], 24, 13, 29);
                                        [1, 3]
```

正如我们所期望的，在无穷大处加点生成了点 P 且返回点 P。

例 4：设 $P = (1,3)$ 为椭圆曲线 $y^2 \equiv x^3 + 24x + 13 \pmod{29}$ 上的点，求 $7P$。

```
> multell ( [1, 3], 7, 24, 13, 29);
                                        [15, 6]
```

例 5：对于上例曲线来说，当 $k = 1,2,3,\ldots,40$ 时，求 $k(1,3)$。

```
> multsell ( [1, 3], 40, 24, 13, 29);
[ [1, 3], [11, 10], [23, 28], [0, 10], [19, 7], [18, 19], [15, 6], [20, 24], [4,
12], [4, 17], [20, 5], [15, 23], [18, 10], [19, 22], [0, 19], [23, 1], [11, 19],
[1, 26], ["infinity","infinity"], [1, 3], [11, 10], [23, 28], [0, 10], [19, 7],
[18, 19], [15, 6], [20, 24], [4, 12], [4, 17], [20, 5], [15, 23], [18, 10], [19,
22], [0, 19], [23, 1], [11, 19], [1, 26], ["infinity","infinity"], [1, 3], [11,
10]]
```

注意，每一个 19 的倍数点是怎样重复的。

例 6：前面的 4 个例子是对素数 29 取模。如果我们用一个可分解的数来计算的话，情形会变得很复杂，因为我们可能在无限地对两个因子进行计算，或者可能是在无限次地对其中一个因子取模，而不理会另一个。因此，如果后面这种情况发生的话就要停止计算，需要扩展一个因子。例如，要计算 $12P$，这里 $P = (1,3)$ 在椭圆曲线 $y^2 \equiv x^3 - 5x + 13 \pmod{209}$ 上：

```
> multell ( [1, 3], 12, -5, 13, 11*19);
```

["factor =", 19]

现在，我们来计算连续的乘积，看看这样下去会发生什么：

```
> multsell ( [1, 3], 12, -5, 13, 11*19);
[ [1, 3], [91, 27], [118, 133], [148, 182], [20, 35], ["factor" =, 19]]
```

当计算到 $6P$ 的时候，结束在无限大 mod 19 处。下面来看看对 209 的两个素数因子（即 19 和 11）取模会发生什么：

```
> multsell ( [1, 3], 12, -5, 13, 19);
[ [1, 3], [15, 8], [4, 0], [15, 11], [1, 16], ["infinity","infinity"],
[1, 3], [15, 8], [4, 0], [15, 11], [1, 16], ["infinity","infinity"]]
> multsell ( [1, 3], 24, -5, 13, 11);
[ [1, 3], [3, 5], [8, 1], [5, 6], [9, 2], [6, 10], [2, 0], [6, 1], [9, 9], [5, 5],
[8, 10], [3, 6], [1, 8], ["infinity","infinity"], [1, 3], [3, 5], [8, 1], [5, 6],
[9, 2], [6, 10], [2, 0], [6, 1], [9, 9], [5, 5]]
```

经历 6 步之后，到达了无限大模 19 的位置，但是用了 14 步才到达无限大模 11 的位置。为了找到 $6P$，我们反转一个数字，用 0 对 19 取模并且非零数对 11 取模。这样做是不可能的，然而它产生了因子 19。这正是椭圆曲线因数分解法的基础。

例 7：使用椭圆曲线法因数分解 193279。

解：首先，需要选择一些任意的椭圆曲线并在每个曲线上找一点。举例来说，取 $P = (2,4)$ 和椭圆曲线

$$y^2 \equiv x^3 - 10x + b \pmod{193279}。$$

为使 P 位于曲线上，我们取 $b = 28$ 。将得到

$$y^2 \equiv x^3 + 11x - 11, P = (1,1)$$

$$y^2 \equiv x^3 + 17x - 14, P = (1,2)。$$

现在计算点 P 的倍数。我们用 $p-1$ 法模拟，因此选择边界 B，记 $B = 12$，并且计算 $B!P$。

```
> multell ( [2, 4], factorial (12), -10, 28, 193279);
                    ["factor =", 347]
> multell ( [1, 1], factorial (12), 11, -11, 193279);
                    [13862, 35249]
> multell ( [1, 2], factorial (12), 17, -14, 193279);
                    ["factor =", 557]
```

让我们更详细地分析在这些例子中发生了什么。

在第一个曲线上，$266P$ 结束在无穷大模 557 处，$35P$ 结束在无穷大模 347 上。因为 $266 = 2 \cdot 7 \cdot 19$，它是一个比 $B = 12$ 大的质因子，所以 $B!P$ 不是无穷大模 557 的，但是 35 可以除尽 $B!$，因此 $B!P$ 是无穷大模 347 的。

在第二个曲线上，$356P$ = 无穷大模 347，$561P$ = 无穷大模 557。因为 $356 = 4 \cdot 89$ 和 $561 = 3 \cdot 11 \cdot 17$，我们不能找到用这个曲线的因数分解。

第三个曲线是令人惊奇的。有 $331P$ = 无穷大模 347 和 $272P$ = 无穷大模 557。由于 331 是素数并且 $272 = 16 \cdot 17$，我们不可能用这个曲线找到因数分解。然而，巧的是在计算 $B!P$ 时中间步骤仍产生了因数分解。下面就是所发生的。在第一步，程序要求增加点（184993，13462）和（20678，150484）。这两个点都是和 557 同余的，但与 347 不同余，因此，倾斜

穿越这二点的线被定义为模 347 的，但却是 0/0 mod 557 的。当我们试着找出分母对 193279 取模的乘积逆元素时，最大公约数算法会产生因子 557，这一现象非常少有。

例 8：这里是如何产生 15.5 节中 ElGamal 加密体制的椭圆曲线的例子。欲知更多的细节，参见原文。椭圆曲线是 $y^2 \equiv x^3 + 3x + 45 \pmod{8831}$，点是 $G = (4, 11)$。艾丽斯的消息是点 $P_m = (5, 1743)$。

鲍勃已经选择了秘密随机数 $a_B = 3$，而且已经计算了 $a_b G$：

```
> multell ( [4, 11], 3, 3, 45, 8831);
                                    [413, 1808]
```

鲍勃公开这个点。艾丽斯选择随机数 $k = 8$，并计算 kG 和 $P_m + k(a_B G)$：

```
> multell ( [4, 11], 8, 3, 45, 8831);
                                    [5415, 6321]
> addell ( [5, 1743], multell ( [413, 1808], 8, 3, 45, 8831), 3, 45, 8831);
                                    [6626, 3576]
```

艾丽斯传送点（5415，6321）和（6626，3576）给鲍勃，她用 a_B 乘以第一个点：

```
> multell ( [5415, 6321], 3, 3, 45, 8831);
                                    [673, 146]
```

然后鲍勃减去艾丽斯发送给他的最后一个点的结果。注意，他所做的减法是通过加第二个被丢弃点来实现的：

```
> addell ( [6626, 3576], [673, -146], 3, 45, 8831);
                                    [5, 1743]
```

因此鲍勃正确地得到了艾丽斯的信息。

例 9：让我们用 15.5 节 DiffieHellman 密钥交换的例子中的方法再产生一些数据：椭圆曲线是 $y^2 \equiv x^3 + x + 7206 \pmod{7211}$，点是 $G = (3, 5)$。艾丽斯选择她的秘密数 $N_A = 12$，鲍勃选择他的秘密数 $N_B = 23$。艾丽斯计算

```
> multell ( [3, 5], 12, 1, 7206, 7211);
                                    [1794, 6375]
```

她发送点（1794，6375）给鲍勃。同时，鲍勃计算

```
multell ( [3, 5], 23, 1, 7206, 7211);
                                    [3861, 1242]
```

并且发送点（3861，1242）给艾丽斯。艾丽斯用 N_A乘以她所收到的值，同时鲍勃用 N_B乘以他所收到的值：

```
> multell ( [3861, 1242], 12, 1, 7206, 7211);
                                    [1472, 2098]
> multell ( [1794, 6375], 23, 1, 7206, 7211);
                                    [1472, 2098]
```

因此，艾丽斯和鲍勃得到了同样的密钥。

MATLAB实例

附录 C

本附录中的例子是为 MATLAB 而写的。如果你有 MATLAB，可以在你的计算机上试试它的一些功能。初用 MATLAB 请看 C.1 节。已经列出了一些函数，可以用 MATLAB 进行实验。在如下地址：

http://www.prenhall.com/washington

可得到和本书配套的 MATLAB 功能。

我们建议你建立一个目录或文件夹来存储这些文件，并且下载它们到这个目录或文件夹。使用这些函数的方法是在存储文件的目录中启动 MATLAB，或启动 MATLAB 并改变现在的目录到保存文件的路径。在 MATLAB 的一些版本中，工作目录可以通过改变命令栏上的当前路径而变更。另外，还可以通过使用 *path* 函数或文件菜单的命令栏上的 Set Path（设置路径）选项增加路径到目录中。

如果没有 MATLAB，仍然可以看这些例子。它们为本书中出现的一些概念提供实例。用于 MATLAB 附录的大部分例子类似于 Mathematica 附录的例子和 Maple 附录的例子。然而 MATLAB 在处理数字时，数字的大小有限制。MATLAB 能正确表示的最大数字是 15 位数字。MATLAB 中的双精度使较大的数字变为近似值。尽管如此，仍然可以使用本书里的大多数例子。

我们认可 MATLAB 的 Maple。这就要求有可用的符号工具箱。MATLAB 的 Maple 在 MATLAB 的学生版本中是不可用的，基于这种情况我们选择类似于 MATLAB 的功能函数，以避免使用符号工具箱。

我们在开始前给出最后一个注释。这在做 MATLAB 练习时要改变显示格式的时候也许是有用的。指令

```
>> format rat
```

设定表示数字的格式。这一计数法对表示大的数字特别有用。传统的 *short*（短）格式在科学计数法中表示大的数字，这种形式经常不显示一些最低（有效）位。

C.1 MATLAB 入门

MATLAB 是工程计算上的一种标准语言。它是一种强有力的语言，快速流行，很快成为数学、自然科学和工程学的标准指导语言。MATLAB 在很多学校都有，一些大学获得了在校园内任何机器上安装 MATLAB 的许可。

为了在个人计算机上启动 MATLAB，双击 MATLAB 图标即可。如果想在 Unix 系统上运行 MATLAB，在提示符后键入 *matlab*。在运行 MATLAB 之前，你将会看到 MATLAB 提示符：

```
>>
```

它提示 MATLAB 正在等候你键入指令。如果想退出 MATLAB，键入 *quit* 即可。

MATLAB 能做基本的算术操作，比如加、减、乘和除。这些可以通过操作符 +，-，* 和 / 来完成。为了得到一个数字的幂，我们用操作符^，来看一个例子：

如果我们键入 2^7 + 125/5 并按 Enter 键

```
>> 2^7 + 125/5
```

MATLAB 将返回答案：

```
ans =
   153
```

在这个例子中，MATLAB 首先运行了求幂，然后除，最后相加得出结果。MATLAB 中运算的次序和我们一直使用的相同。我们也能用圆括号改变 MATLAB 中的运算次序。下面的例子体现了这一点：

```
>> 11 * ((128/(9+7) - 2^(72/12)))
ans =
  -616
```

在例子中，MATLAB 已经得出计算结果 *ans*，*ans* 是 MATLAB 用来储存计算结果的变量。把计算的结果分配给一个特定的变量是可以的。例如，

```
>> spot =17
spot =
    17
```

就是分配值 17 给变量 *spot*。在计算中使用变量也是可以的：

```
>> dog =11
dog =
    11
>> cat =7
cat =
    7
>> animals =dog +cat
animals =
    18
```

由于 MATLAB 有许多有用的计算功能，所以 MATLAB 也可以像一个先进的科学计算器一样操作。例如，我们使用 *sqrt* 函数能得到一个数的平方根，比如下面这个例子：

```
>> sqrt(1024)
ans =
    32
```

MATLAB 还有很多实用的函数。*mod*，*factorial*，*factor*，*prod* 和 *size* 都是本书中很有用的函数。

在 MATLAB 中有帮助功能。如果使用个人计算机，可以键人 *help*，或使用帮助菜单。如果使用的是 Unix 系统，帮助的使用基本上和个人计算机相同。然而，如果你使用的是 6.0 版本前的 MATLAB 版本，那么不能使用下拉式菜单。MATLAB 也可以通过在指令行键人 *help commandname* 以得到帮助。例如，要得到我们经常用到的 mod 函数的帮助，键人：

```
>> help mod
```

MATLAB 有一个可用的工具箱。工具箱由许多实现特殊应用任务的功能集所组成。例如，最优化工具箱（Optimization toolbox）提供了实现线性和非线性最优化的功能集。通常，不是所有的工具箱都可用。然而，对于我们的目的而言，由于我们只需要使用一般的 MATLAB 功能集，并且已经建造了我们自己的关于密码背后的数字理论的功能集，因此这已经不是问题了。

用于 MATLAB 的基本数据类型是矩阵。MATLAB 被设计为使用矩阵和向量作为最基本的数据类型。由于许多数学的和科学的问题都要用到矩阵和向量，因此这种设计是很自然的。

我们来看怎样在 MATLAB 中输人一个矩阵的例子。假如想输人如下这个矩阵：

$$A=\begin{bmatrix}1&1&1&1\\1&2&4&8\\1&3&9&27\\1&4&16&64\end{bmatrix}$$

我们在指令行键人：

```
>> A = [1 1 1 1; 1 2 4 8; 1 3 9 27; 1 4 16 64]
```

MATLAB 立即返回

```
A =
    1  1  1  1
    1  2  4  8
    1  3  9  27
    1  4  16  64
```

当输人矩阵或向量的时候，需要用到一些基本的规则。首先，矩阵和向量是以 [开始的，以] 结束的。其次，空格或逗号分开行中的元素。一个分号结束每行。最后，我们可以在结尾处输人一个分号，以阻止 MATLAB 显示指令的输出。

为了定义行向量，使用空格或逗号。例如，

```
>> x = [2,4,6,8,10,12]
x =
     2   4   6   8   10   12
```

为了定义列向量，使用分号。例如，

```
>> y = [1; 3; 5; 7]
y =
     1
     3
     5
```

```
    7
```

为了存取 y 的某个元素。把所要的索引放入括弧。例如，y (1) = 1，y (2) = 3，等等。

MATLAB 提供同时存取多个元素的有用的记号法。例如，要存取 x 的第三、第四和第五个元素，我们只需键入

```
>> x (3: 5)
ans =
    6   8   10
```

3:5 指示 MATLAB 始于 3 终于 5。为了间隔地存取 x 中的元素，需要这样做

```
>> x (1: 2: 6)
ans =
    2   6   10
```

我们也可以这样对数组操作。例如，

```
>> A (1: 2: 4, 2: 2: 4);
ans =
    1   1
    3   27
```

记号 1: n 也可以用来定义一个变量。例如，

```
>> x =1: 7
```

返回

```
x =
  1   2   3   4   5   6   7
```

MATLAB 提供确定向量或矩阵变量维数的 *size* 函数。例如，如果我们想在输入矩阵 A 之前知道其维数，那么可以这样做

```
>> size (A)
ans =
   4   4
```

以不同的格式显示数字经常是必需的。MATLAB 为显示计算结果提供了一些输出格式。为了得到可用的格式类型，键入

```
>> help format
```

short 格式是默认格式，对做许多计算非常方便，然而，在本书中，我们使用全部用长格式的数字，而 *short* 格式将会切断长数字后面的数字。例如，

```
>> a =1234567899
a =
    1.2346e +009
```

不用 *short* 格式，我们将用 *rational*(有理数) 格式。转变 MATLAB 到使用有理数的格式，键入

```
>> format rat
```

作为一个例子，如果我们举与前面相同的例子，此时就得到了一个不同的结果：

```
>> a =1234567899
a =
    1234567899
```

因为它允许我们表示微小形式的分数，所以这个格式是很有用的，举例来说，

```
>> 111/323
ans =
   111/323
```

在许多情形中，压缩计算的结果会很方便。为了使 MATLAB 压缩打印结果，指令后一定要跟一个分号。还有，很多指令可能在同一行输入，中间由逗号分开。例如，

```
>> dogs =11, cats =7; elephants =3, zebras =19;
dogs =
    11
elephants =
     3
```

返回变量 *dogs* 和 *elephants* 的值，但不显示 *cats* 和 *zebras* 的值。

MATLAB 也能处理由文本构成的变量。字符串被看作是符号数组。要给字符串分配一个变量，用单引号标记文本。例如，

```
>> txt = 'How are you today?'
```

返回

```
txt =
    How are you today?
```

字符串和向量一样是有大小的。例如，变量 txt 的大小被给定

```
>> size(txt)
ans =
    1   18
```

逐个编辑符号也是可以的。例如，下面这个指令就改变了 txt 的第一个单词：

```
>> txt(1) = 'W'; txt(2) = 'h'; txt(3) = 'o'
txt =
   Who are you today?
```

当你在 MATLAB 中工作的时候，它会提醒你要给你输入的变量赋予一定的值。回顾以前的指令，按向上键和向下键。要看你创建的变量，在提示行键入 *who*。类似的指令 *whos* 能给出变量的大小和类型的相关信息。

注意：

1. 要使用例子中的指令，应该从网站 *http://www.prenhall.com/washington* 下载的文件目录中运行 MATLAB。

2. 一些例子和上机题使用长密文，等等。为方便起见，这些密文存储在文件 ciphertexts.m 中，可在提示行键入 *ciphertexts* 提取，然后可由名字提取密码。举例来说，请看第 2 章上机题例 4。

C.2 第 2 章实例

例 1： 使用移位密码获得密文 kddkmu，尝试所有的可能性来解密它。

```
>> allshift (' kddkmu')

kddkmu
leelnv
mffmow
nggnpx
ohhoqy
piiprz
qjjqsa
rkkrtb
sllsuc
tmmtvd
unnuwe
voovxf
wppwyg
xqqxzh
yrryai
zsszbj
attack
buubdl
cvvcem
dwwdfn
exxego
fyyfhp
gzzgiq
haahjr
ibbiks
jccjlt
```

如你所见，attack 是在列表上出现的惟一单词，所以它就是明文。

例 2：运用仿射函数 $7x+8$ 加密明文信息 cleopatra：

```
>> affinecrypt (' cleopatra' , 7, 8)
ans =
whkcjilxi
```

例 3：密文 mzdvezc 是用仿射函数 $5x+12 \pmod{26}$ 加密的。对它进行解密。

解：首先，解关于 x 的方程 $y \equiv 5x+12 \pmod{26}$，得到 $x \equiv 5^{-1}(y-12)$。我们需要找到 $5 \pmod{26}$ 的逆：

```
>> powermod (5, -1, 26)
ans =
  21
```

因此，$x \equiv 21(y-12) \equiv 21y-12 \cdot 21$。改变 $-12 \cdot 21$ 为标准格式：

```
>> mod ( -12*21, 26)
ans =
```

```
8
```

因此，解密函数是 $x = 21y + 8$。为了破解这个信息：

```
> > affinecrypt ('mzdvezc ', 21, 8)
ans =
anthony
```

你可能觉得奇怪，明文是按下面的方式被加密的：

```
> > affinecrypt ('anthony' , 5, 12)
ans =
mzdvezc
```

例 4：这是一个关于文本的 Vigenère 密码的例子。我们来看看如何获得 2.3 节中破解密码时的数据。在文件 ciphertexts. m 中，密文以 *vvhq* 为名被存储。如果你还没有做，装载文件 ciphertexts. m：

```
> > ciphertexts
```

现在能用变量 vvhq 来获得密文：

```
> > vvhq
vvhqwvvrhmusgjgthkihtssejchlsfcbgvwcrlryqtfsvgahwkcuhwauglq
hnslrljshbltspisprdxljsveeghlqwkasskuwepwqtwvspgoelkcqyfnsv
wljsniqkgnrgybwlwgoviokhkazkqkxzgyhcecmeiujoqkwfwvefqhkijrc
lrlkbienqfrjljsdhgrhlsfqtwlauqrhwdmwlgusgikkflryvcwvspgpmlk
assjvoqxeggveyggzmljcxxljsvpaivwikvrdrygfrjljslveggveyggeia
puuisfpbtgnwwmuczrvtwglrwugumnczvile
```

现在在密文中找出字母的频率。使用 *frequency* 函数。指令 *frequency* 自动显示字母及其计数，因此我们放一个分号以在指令结束的时候阻止 MATLAB 显示计数两次。

```
> > fr = frequency (vvhq);

a  8
b  5
c 12
d  4
e 15
f 10
g 27
h 16
i 13
j 14
k 17
l 25
m  7
n  7
o  5
p  9
q 14
```

```
r  17
s  24
t   8
u  12
v  22
w  22
x   5
y   8
z   5
```

我们来计算移动 1，2，3，4，5，6 位后的相同数：

```
>> coinc (vvhq, 1)
ans =
    14
>> coinc (vvhq, 2)
ans =
    14
>> coinc (vvhq, 3)
ans =
    16
>> coinc (vvhq, 4)
ans =
    14
>> coinc (vvhq, 5)
ans =
    24
>> coinc (vvhq, 6)
ans =
    12
```

可以得出密钥的长度大约是 5。我们来看看第 1，第 6，第 11，... 个字母（即，适合 I mod 5 位置的字母）。*choose* 函数能完成这些。函数 *choose*(*txt*，*m*，*n*) 能找出 txt 中适合 n mod m 位置的每个字母。

```
>> choose (vvhq, 5, 1)
ans =
vvuttcccqgcunjtpjgkuqpknjkygkkgcjfqrkqjrqudukvpkvggjjivgjggpfncwuce
```

现在我们对前面的子串做频率计数。为此，需要使用 *frequency* 函数而且以 ans 作为输入。MATLAB 中，如果一个指令没有输出结果就被运行，MATLAB 将会把输出放入变量 ans 中。

```
>> frequency (ans);

a  0
b  0
c  7
d  1
```

```
e  1
f  2
g  9
h  0
i  1
j  8
k  8
l  0
m  0
n  3
o  0
p  4
q  5
r  2
s  0
t  3
u  6
v  5
w  1
x  0
y  1
z  0
```

为了将其表达为频率的向量，我们使用 *vigvec* 函数。*vigvec* 函数不仅仅能显示频率计数，而且还会返回一个包含频率的向量。在下列各项的输出中，由于我们已经限制了频率计数的格式，故它们均以 *short* 格式显示结果。

```
>> vigvec (vvhq, 5, 1)

ans =
    0
    0
    0.1045
    0.0149
    0.0149
    0.0299
    0.1343
    0
    0.0149
    0.1194
    0.1194
    0
    0
    0.0448
```

```
0
0.0597
0.0746
0.0299
0
0.0448
0.0896
0.0746
0.0149
0
0.0149
0
```

这个向量的点积和字母频率向量的变化如下进行计算:

```
>> corr (ans)
 ans =
     0.0250
     0.0391
     0.0713
     0.0388
     0.0275
     0.0380
     0.0512
     0.0301
     0.0325
     0.0430
     0.0338
     0.0299
     0.0343
     0.0446
     0.0356
     0.0402
     0.0434
     0.0502
     0.0392
     0.0296
     0.0326
     0.0392
     0.0366
     0.0316
     0.0488
     0.0349
```

第三项是最大值，但有时找出最大值是很难的。一种找出它的方法是

```
> > max (ans)
ans =
   0.0713
```

现在就很容易通览记录找到这个数字（它通常只出现一次）。由于它在第三个位置出现，第一个 Vigenère 密码变换是移动了 2 位，对应字母 *c*。与刚才所用相似的程序（使用 *vigvec* (*vvhq*, 5, 2), . . . , *vigvec*(*vvhq*, 5, 5)）显示其他的移位可能是 14，3，4，18。我们检查一下解密得到的密钥。

```
> > vigenere (vvhq, - [2, 14, 3, 4, 18] )
ans =
themethodusedforthepreparationandreadingofcodemessagesissim
pleintheextremeandatthesametimeimpossibleoftranslationunles
sthekeyisknowntheeasewithwhichthekeymaybechangedisanotherpo
intinfavoroftheadoptionofthiscodebythosedesiringtotransmiti
mportantmessageswithouttheslightestdangeroftheirmessagesbei
ngreadbypoliticalorbusinessrivalsetc
```

记录中，明文最初是用如下指令加密的

```
> > vigenere (ans, [2, 14, 3, 4, 18] )
ans =
vvhqwvvrhmusgjgthkihtssejchlsfcbgvwcrlryqtfsvgahwkcuhwauglq
hnslrljshbltspisprdxljsveeghlqwkasskuwepwqtwvspgoelkcqyfnsv
wljsniqkgnrgybwlwgoviokhkazkqkxzgyhcecmeiujoqkwfwvefqhkijrc
lrlkbienqfrjljsdhgrhlsfqtwlaugrhwdmwlgusgikkflryvcwvspgpmlk
assjvoqxeggveyggzmljcxxljsvpaivwikvrdrygfrjljslveggveyggeia
puuisfpbtgnwwmuczrvtwglrwugumnczvile
```

例 5：密文

$$22,09,00,12,03,01,10,03,04,08,01,17$$

是使用如下矩阵的希尔密码加密的：

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 10 \end{pmatrix}$$

试破解它。

解：一个矩阵 $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ 是以 $[a,b;c,d]$ 的方式输入的。键入 $M*N$ 得出矩阵 M 和 N 的乘积矩阵。键入 $v*\ M$ 实现在向量 v 的右边乘上矩阵 M。

首先，我们把上述矩阵放入变量 M 。

```
> > M = [1 2 3; 4 5 6; 7 8 10]
   M =
     1   2   3
     4   5   6
     7   8   10
```

然后，求逆矩阵并 mod 26：

```
> > Minv = inv (M)
Minv =
        -2/3    -4/3    1
        -2/3    11/3    -2
         1      -2      1
```

一旦执行 mod 26，就不能在像 2/ 3 这样的数字前停止。需要免除分母并且减少 mod 26 。为此，我们乘 3 以去掉分数的分子，然后乘 3 mod 26 的转置把“分母”放回原处（见 3. 3 节）：

```
> > M1 = (Minv * 3)
M1 -
     -2   -4   3
     -2   11  -6
      3   -6   3
> > M2 = round (mod (M1 * 9, 26))
M2 =
      8    16    1
      8    21    24
      1    24    1
```

注意，在计算 M2 时用了函数 *round*。这是因为 MATLAB 使用了浮点来完成运算过程，并且计算逆矩阵 *Minv* 产生了与整个数字稍微不同的数字。*M2* 是矩阵 *M* mod 26 的逆矩阵。能按如下方法检验：

```
> > mod (M2 * M, 26)
ans =
     1   0   0
     0   1   0
     0   0   1
```

为了破解，将密文每 3 个数字分成一组，并在每组的右边乘上刚才所得的逆矩阵：

```
> > mod ( [22, 9, 0] * M2, 26)
ans =
     14   21   4
> > mod ( [12, 3, 1] * M2, 26)
ans =
     17   19   7
> > mod ( [10, 3, 4] * M2, 26)
ans =
    4   7   8
> > mod ( [8, 1, 17] * M2, 26)
ans =
    11   11   23
```

因此，明文是 14, 21, 4, 17, 19, 7, 4, 7, 8, 11, 11, 23。可以还原回原来的字母：

```
> > int2text ( [14 21 4 17 19 7 4 7 8 11 11 23])
ans =
```

```
overthehillx
```

注意，最后的 x 被附加到明文后是为了要构成3个字母的分组。

例6：计算递归式子 $x_{n+5} \equiv x_n + x_{n+2} \pmod 2$ 的前50项。初始值为0，1，0，0，0。

解：系数的向量是{1，0，1，0，0}，初始向量是{0，1，0，0，0}。键入

```
>> lfsr ( [1 0 1 0 0], [0 1 0 0 0], 50)
ans =
    Columns 1 through 12
    0   1   0   0   0   0   1   0   0   1   0   1
    Columns 13 through 24
    1   0   0   1   1   1   1   1   0   0   0   1
    Columns 25 through 36
    1   0   1   1   1   0   1   0   1   0   0   0
    Columns 37 through 48
    0   1   0   0   1   0   1   1   0   0   1   1
    Columns 49 through 50
    1   1
```

例7：假设LFSR序列的前20项是1，0，1，0，1，1，1，0，0，0，0，1，1，1，0，1，0，1，0，1。找出一个产生此序列的递归式。

解：首先，我们定义递归运算的长度。指令 *lfsrlength* [v，n] 用于计算在2.11节中出现的前 n 个阵列的模2行列式。最后的非零行列式给出了递归的长度。

```
>> lfsrlength ( [1 0 1 0 1 1 1 0 0 0 0 1 1 1 0 1 0 1 0 1], 10)
    Order  Determinant
      1          1
      2          1
      3          0
      4          1
      5          0
      6          1
      7          0
      8          0
      9          0
      10         0
```

最后的非零行列式是第6个，故我们猜这个递归的长度是6。为了找出这个系数：

```
>> lfsrsolve ( [1 0 1 0 1 1 1 0 0 0 0 1 1 1 0 1 0 1 0 1], 6)
ans =
    1  0  1  1  1  0
```

得到递归式如下

$$x_{n+6} \equiv x_n + x_{n+2} + x_{n+3} + x_{n+4} \pmod 2)\ 。$$

例8：密文0，1，1，0，1，0，1，0，1，0，0，1，1，0，0，0，1，0，1，0，1，0，1，0，1，0，1，0，1，0，0，1，0，0，0，1，0，1，1，0是由明文对2取模再加上LFSR的输出得到的（即，用明文同LFSR的输出进行异或）。假设知道明文是以1，1，1，1，1，

1，0，0，0，0，0，0，1，1，1，0，0 开始的，找出余下明文的内容。

解：利用密文和已知的明文部分异或，得到 LFSR 输出的开始部分：

```
>> x=mod([1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 0 0] + [0 1 1 0 1 0 1 0 1 0 0 1 1 0 0 0 1], 2)
x =
    Columns 1 through 12
      1  0  0  1  0  1  1  0  1  0  0  1
    Columns 13 through 17
      0  1  1  0  1
```

这是 LFSR 输出的开始部分。现在我们来找出这个递归的长度：

```
>> lfsrlength(x, 8)
    Order  Determinant
      1        1
      2        0
      3        1
      4        0
      5        1
      6        0
      7        0
      8        0
```

我们猜长度是 5。为找出这个系数：

```
>> lfsrsolve(x, 5)
ans =
    1  1  0  0  1
```

现在使用刚刚得出的系数再加上 LFSR 输出的最初五项，可以得出 LFSR 的完全输出：

```
>> lfsr([1 1 0 0 1], [1 0 0 1 0], 40)
ans =
    Columns 1 through 12
    1  0  0  1  0  1  1  0  1  0  0  1
    Columns 13 through 24
    0  1  1  0  1  0  0  1  0  1  1  0
    Columns 25 through 36
    1  0  0  1  0  1  1  0  1  0  0  1
    Columns 37 through 40
    0  1  1  0
```

当我们将 LFSR 的输出和密文异或，可以回到明文：

```
>> mod(ans + [0 1 1 0 1 0 1 0 1 0 0 1 1 0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0 1 0 0 0 1 0 1 1 0], 2)
ans =
   Columns 1 through 12
    1  1  1  1  1  1  0  0  0  0  0  0
   Columns 13 through 24
    1  1  1  0  0  0  1  1  1  1  0  0
```

```
Columns 25 through 36
 0  0  1  1  1  1  1  1  1  0  0  0
Columns 37 through 40
 0  0  0  0
```

这就是明文。

C.3 第3章实例

例 1：找出 gcd(23456, 987654)。

```
>> gcd (23456, 987654)
ans =
    2
```

例 2：在整数范围内求 x，y，使之满足 $23456x+987654y=2$。

```
>> [a, b, c] =gcd (23456, 987654)
a =
    2
b =
    -3158
c =
    75
```

这就意味着 2 是最大公约数，并且 $23456\cdot(-3158)+987654\cdot 75=2$。

例 3：计算 $234\cdot 456 \pmod{789}$。

```
>> mod (234*456, 789)
ans =
    189
```

例 4：计算 $234^{567} \pmod{9871}$。

```
>> powermod (234, 567, 9871)
ans =
    5334
```

例 5：计算 8787(mod 91919) 的乘法逆元素。

```
>> powermod (8787, -1, 91919)
ans =
    71374
```

例 6：求解 $7654x\equiv 2389 \pmod{65537}$。

解：为了解决这个问题，我们效仿 3.3 节中描述的方法。先计算 7654^{-1}然后用 2389 乘之：

```
>> powermod (7654, -1, 65537)
ans =
    54637
>> mod (ans*2389, 65537)
```

```
ans =
   43626
```

例 7：找出满足下列条件的 x：

$$x \equiv 2(\bmod 78), \quad x \equiv 5(\bmod 97), \quad x \equiv 1(\bmod 119)。$$

解：为了解决这个问题，我们使用 *crt* 函数。

```
>> crt ( [2 5 1], [78 97 119])
ans =
   647480
```

可以检验这个结果：

```
>> mod (647480, [78 97 119])
ans =
     2   5   1
```

例 8：把 123450 因数分解为素数。

```
>> factor (123450)
ans =
     2   3   5   5   823
```

即 $123450 = 2^1 3^1 5^2 823^1$。

例 9：求 $\Phi(12345)$ 的值。

```
>> eulerphi (12345)
ans =
    6576
```

例 10：找出素数 65537 的本原根。

```
>>primitiveroot (65537)
ans =
   3
```

因此，3 是素数 65537 的本原根。

例 11：找出矩阵 $\begin{pmatrix} 13 & 12 & 35 \\ 41 & 53 & 62 \\ 71 & 68 & 10 \end{pmatrix}(\bmod 999)$ 的逆矩阵。

解：首先，把矩阵赋给 M 并输入。

```
>>M= [13 12 35; 41 53 62; 71 68 10];
```

接着，在求模前对矩阵求逆：

```
>> Minv=inv (M)
Minv =
    233/2158     -539/8142     103/3165
    -270/2309    139/2015      -40/2171
    209/7318     32/34139      -197/34139
```

为了从 *Minv* 的数字中去掉分数，我们需要乘上 M 的行列式，然后需要乘上行列式模 999 的逆。

```
>> Mdet=det (M)
```

```
Mdet =
   -34139
>>powermod(Mdet, -1, 999)
ans =
   589
```

得出解决方法

```
>>mod(Minv*589*Mdet,999)
ans =
     772    472    965
     641    516    851
     150    133    149
```

因此，逆矩阵模 999 为 $\begin{pmatrix} 772 & 472 & 965 \\ 641 & 516 & 851 \\ 150 & 133 & 149 \end{pmatrix}$。

在许多情况下，通过观察法决定要去掉的公分母是可能的。当不是这种情况时，注意原始矩阵的行列式总是以一个公分母的形式存在的。

在这个例子中，我们用矩阵的行列式作为要去掉的公分母。原始矩阵的行列式总是以一个公分母的形式存在的。

例 12：求出 29887 模素数 $p=32579$ 的一个平方根。

解：由于 $p\equiv 3 \pmod 4$，我们可以使用 3.9 节中的方法：

```
>>powermod(29887,(32579+1)/4, 32579)
ans =
   19237
```

另外一个平方根是它的负数：

```
>>mod(-ans,32579)
ans =
   13342
```

例 13：令 $n=34222273=9803.3491$，求出 $x^2=19101358 \pmod{34222273}$ 的所有 4 个解。

解：首先，找一个平方根分别对两个素数因子取模，两者都与 $3 \pmod 4$ 同余：

```
>>powermod(19101358,(9803+1)/4, 9803)
  ans =
     3998
>>powermod(19101358,(3491+1)/4, 3491)
  ans =
     1318
```

因此，这些平方根与 $\pm 3998 \pmod{9803}$ 和 $\pm 1318 \pmod{3491}$ 同余。利用中国剩余定理，有 4 种方法来组合这些：

```
>>crt([3998 1318],[9803 3491])
ans =
   43210
>>crt([-3998 1318],[9803 3491])
```

```
ans =
   8397173
>> crt([3998 -1318],[9803 3491])
ans =
   25825100
>> crt([-3998 -1318],[9803 3491])
ans =
   34179063
```

这就是所要求的 4 个平方根。

C.4　第6章实例

例 1：如前面所指出的，MATLAB 在处理数字的大小方面是受限制的。MATLAB 所能精确表示的最大数是 10^{15}。MATLAB 中用的双精度数可以近似表示更大的数字。然而，我们仍然可以用 MATLAB 产生小于 10^{7} 的素数。*nextprime* 和 *randprime* 这两个函数已经被写出来产生素数。函数 *nextprime* 以一个数 n 作为输入，并尝试找到 n 后面的下一个素数。函数 *randprime* 以数 n 作为输入，并尝试找到 1 到 n 之间的一个素数。这两个函数使用第 6 章中描述的 Miller-Rabin 测试法。

```
>> nextprime(346735)
ans =
   346739
>> randprime(888888)
ans =
   737309
```

例 2：假设你想把文本 hello 变换成数字：

```
>> text2int1('hello')
ans =
   805121215
```

注意我们用的是 $a=1, b=2, \ldots, z=26$，否则 a 在信息首部时将消失。（一个更高效的方法建立在基数 27 之上，即信息的数字形式为 $8+5\cdot 27+12\cdot 27^{2}+12\cdot 27^{3}+15\cdot 27^{4}=1497902$。注意这种方法使用更少的位数。）现在假如你想把它转换回字母：

```
>> int2text1(805121215)
ans =
   hello
```

例 3：在 $n=823091$，$e=17$ 的条件下用 RSA 法加密信息 hi。

解：首先把信息转换成数字：

```
>> text2int1('hi')
ans =
   809
```

现在，把它提升到 e 次幂对 n 取模：

```
> > powermod (ans, 17, 823091)
ans =
   596912
```

例 4：对上例的密文解密。

解：首先，我们要找出解密指数 d。为此，需要找到 $\Phi(823091)$。一种方法是

```
> > eulerphi (823091)
ans =
   821184
```

另一种方法是把 n 分解成 $p \cdot q$ 的形式，然后计算 $(p-1)(q-1)$：

```
> > factor (823091)
ans =
    659   1249
> > 658 * 1248
ans =
   821184
```

由于 $de \equiv 1(\bmod \phi(n))$，我们进行下面的计算（注意，我们正在找的是 $e \bmod \phi(n)$，而不是 mod n 的逆）：

```
> > powermod (17, -1, 821184)
ans =
   48305
```

因此，$d=48305$。为了解密，提高密文到 d 次幂模 n：

```
> > powermod (596912, 48305, 823091)
ans =
   809
```

最后，转换回字母：

```
> > int2text1 (ans)
ans =
   hi
```

例 5：令 $n=823091$ 和 $e=17$，用 RSA 加密 sunshine。

解：首先，把明文信息转换为数字：

```
> > text2int1 ('sunshine')
ans =
   1921141908091405
```

假设我们简单地把它提高到 e 次幂模 n：

```
> > powermod (ans, 17, 823091)
ans =
   640791
```

如果我们解密（从例 4 中得出 d），可以得到

```
> > powermod (ans, 48305, 823091)
ans =
   340339
```

这不是原来的明文。原因是明文比 n 大，因此我们已经得到了明文模 n：

```
>> mod (text2int1 ('sunshine'), 823091)
ans =
   340339
```

我们需要把明文分组，每组小于 n。在本例中，我们一次使用 3 个字母：

192114 190809 1405

```
>> powermod (192114, 17, 823091)
ans =
   686022
>> powermod (190809, 17, 823091)
ans =
   660591
>> powermod (1405, 17, 823091)
ans =
   702126
```

因此密文为 686022660591702126。注意：这个密文无法还原回字母。实际上，它不与任何字母文本相对应。

分别解密每一组：

```
>> powermod (686022, 48305, 823091)
ans =
   192114
>> powermod (660591, 48305, 823091)
ans =
   190809
ans =
   1405
```

例 6：令素数 $p=857$，$q=683$，加密指数 $e=9007$，用 RSA 加密法给 bat，cat 和 hat 加密。

解：首先，我们输入变量 p，q 和 e。

```
>> p=857; q=683; e=9007;
```

为计算 n，输入指令

```
>> n=p*q;
```

通过下式可计算出密文：

```
>> powermod (text2int1 ('bat'), e, n)
ans =
   54984
>> powermod (text2int1 ('cat'), e, n)
ans =
   236057
>> powermod (text2int1 ('hat'), e, n)
ans =
   382934
```

例 7：在上面的例子中，令 $e=9007$ 和 $n=585331$。对于 e 和 n 的这个选择，相应的解密指数为 $d=265743$。（怎样能计算出呢？）我们用 d 和 e 来因数分解 n。

解决方法：利用 6.4 节中通用的指数因数分解法。首先，定义 $y=ed-1$，并再令 $y=2^s m$。一种方法是首先计算 y，然后持续用 2 除直到得到一个奇数。

```
>> y = e*d-1
y =
    2393547200
>> y/2
ans =
   1196773600
>> ans/2
ans =
   598386800
```

继续用这种方法直到得到 37399175。定义 $m=37399175$。选择一随机整数 a。希望有好运，我们取 13。与通用的指数因数分解法一样，我们计算

```
>> powermod (13, m, n)
ans =
    530690
```

由于这个数不等于 $\pm 1 \pmod n$，继续乘方直至得到 ± 1：

```
>> powermod (ans, 2, n)
ans =
   450781
>> powermod (ans, 2, n)
ans =
   1
```

由于在 1 前的最后一个数字不是 $\pm 1 \pmod n$，我们有一个 $x \not\equiv \pm 1 \pmod n$ 的例子，其中 $x^2 \equiv 1$。因此，$\gcd(x-1, n)$ 是 n 的一个非平凡因子：

```
>> gcd (450781 - 1, n)
ans =
   683
```

这是因子 q。通过 n/q，我们可以计算另外一个因子。

```
>> n/ans
ans =
   857
```

由于 MATLAB 本身不能处理大的数字，我们将跳过 6.5 节中讨论的 RSA 挑战的例子。在 Mathematica 和 Maple 计算机例子中我们介绍过 RSA 挑战的例子。

现在我们为那些有符号工具包的读者示范一下如何在 MATLAB 中执行 Maple 的一些指令。

首先，计算 $234567^{876543} \pmod{565656565}$，输入

```
>> maple ('234567&^876543 mod 565656565')
ans =
```

```
    473011223
```

为计算 574786324786343457 后的下一个素数，输入

```
>> maple ('nextprime (574786324786343457)')
ans =
   574786324786343459
```

关于 Maple 的其他一些有用的指令，我们建议读者参考附录 B 中的 Maple 例子。

C.5 第 8 章实例

例 1：假设一个房间有 23 个人，那么至少有两个人生日相同的概率是多少?

解：没有两个人生日相同的概率是 $\Pi_{i=1}^{22}(1-i/365)$（注意，乘积在 $i=22$ 时结束，而不是 $i=23$ 时）。从 1 中减去这个概率就是至少有两个人生日相同的概率：

```
>> 1-prod (1 - (1: 22) /365)
    ans =
    0.5073
```

例 2：假设一位懒惰的电话公司职员通过随机选择 7 位数字来分配电话号码。在一个有 10000 部电话的城市，两个人分到同一个号码的概率是多少?

```
>> 1-prod (1 - (1: 9999)/10^7)
   ans =
        0.9933
```

注：电话号码大约是可能电话数量的平方根的 3 倍。这就意味着概率非常高。由 8.4 节，我们推测如果有 $\sqrt{2(\ln 2)10^7}\approx 3723$ 个左右的电话，就有 50% 的号码相同的机会。下而来验证一下它的正确性：

```
>> 1-prod (1 - (1: 3722)/10^7)
ans =
    0.4999
```

C.6 第 10 章实例

例 1：假设有一个（5，8）Shamir 秘密共享方案，任何一个方案都是对素数 $p=987541$ 求模的。5 个共享方案是

(9853, 853), (4421, 4387), (6543, 1234), (93293, 78428), (12398, 7563),

试找到这个密秘。

解：函数 *interppoly*（x，f，m）计算通过点（x_i，f_i）的插值多项式，该算法以 m 为模进行计算。

为了使用这个函数，需要引入一个包含 x 值的向量和另外一个包含共享值的向量。用下面两条指令可实现：

```
>> x = [9853 4421 6543 93293 12398];
```

```
>> s = [853 4387 1234 78428 7563];
```

现在我们计算这个插值多项式的系数。

```
>> y = interppoly (x, s, 987541)
y =
     678987   14728   1651   574413   456741
```

第一个值与多项式的常数项相关，是密秘值。因此，678987 是密秘。

C.7 第 11 章实例

例 1：这里有一个你可以玩的游戏。它本质上是 11.2 节中通过电话玩扑克牌的简单版本。这里有 5 张牌：ten，jack，queen，king，ace。把它们缩写成：ten，ace，que，jac，kin。洗牌后通过把它们的数字提高到一个随机指数模 300649 来伪装它们的值，猜测哪一个是 ace。

首先，输入牌并按下列步骤把它们转换成数字：

```
>> cards = ['ten' ;'ace'; 'que'; 'jac'; 'kin'];
>> cvals = text2int1 (cards)
cvals =
   200514
   10305
   172105
   100103
   110914
```

然后，我们选任意的指数 k，它将会被用于隐藏操作。为了使我们不能作弊和看到正在使用的 k 值，在函数 *khide* 后加了个分号。

```
>> p = 300649;
>> k = khide (p);
```

现在，对伪装后的牌洗牌（它们的数字被提高到 k 次幂模 p）：

```
>> shufvals = shuffle (cvals , k, p)
shufvals =
    226536
    226058
    241033
    281258
    116809
```

就是这 5 张牌。没有一张像是 ace；那是因为它们的数字已经被提高到幂次方对素数取模。随便猜一下，让我们看看你是否正确。

```
>> reveal (shufvals, k, p)
ans =

jac
que
```

```
ten
kin
ace
```

我们再来一次：

```
>> k = khide (p);
>> shufvals = shuffle (cvals, k, p)
shufvals =
     117135
     144487
     108150
     266322
     264045
```

做出你的选择（注意，因为用了不同的随机指数，这些数字是不同的）。你是否很幸运？

```
>> reveal (shufvals, k, p)
ans =
kin
jac
ten
que
ace
```

也许你需要一些帮助。让我们再玩一次：

```
>> k = khide (p);
>> shufvals = shuffle (cvals, k, p)
shufvals =
    108150
    144487
    266322
    264045
    117135
```

现在我们请求提示：

```
>> advise (shufvals, p);
Ace Index:   4
```

提示我们第四张是 ace。让我们来看看：

```
>> reveal (shufvals, k, p)
ans =

ten
jac
que
ace
kin
```

这是怎么得到的？看看 11.2 节中关于“怎样作弊”的那部分。注意，如果提高牌的数字到

$(p-1)/2$ 次幂模 p，可得

```
>> powermod (cvals, (p-1)/2, p)
ans =
        1
   300648
        1
        1
        1
```

因此，只有 ace 是模 p 的二次非剩余。

C.8 第15章实例

例 1：我们想画椭圆曲线 $y^2 = x(x-1)(x+1)$。首先，建立一个包含要画的曲线方程的字符串 v。

```
>> v='y^2 - x* (x-1) * (x+1)';
```

然后用 *ezplot* 指令分割椭圆曲线。

```
>> ezplot (v,[-1, 3, -5, 5])
```

划分的形状如图 C.1 所示。在前面的指令中使用 [-1, 3, -5, 5] 来定义 x 轴和 y 轴的范围。

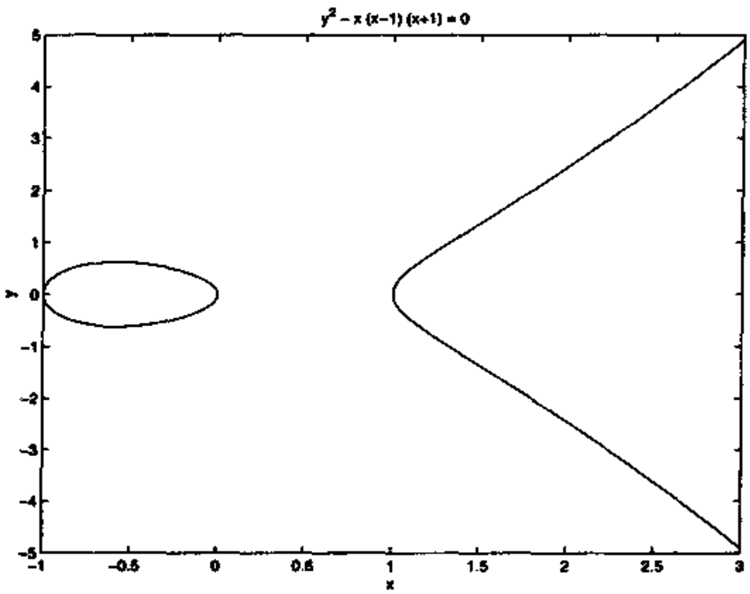


图 C.1 椭圆曲线 $y^2 = x(x-1)(x+1)$ 的形状

例 2：在椭圆曲线 $y^2 \equiv x^3 + 24x + 13 \pmod{29}$ 上添加点 (1, 3) 和 (3, 5)。

```
>> addell ( [1, 3],[3, 5] , 24, 13, 29)
ans =
    26   1
```

可以检验点 (26, 1) 是否在曲线 $26^3 + 24 \cdot 26 + 13 \equiv 1^2 \pmod{29}$ 上。

例 3：在前面例子中曲线的无穷大处添加点（1，3）。

```
>> addell ([1, 3],[inf, inf], 24, 13, 29)
ans =
    1   3
```

正如我们所预期的，在无穷大处添加了点 P 则返回点 P。

例 4：让 $P = (1, 3)$ 成为曲线 $y^2 \equiv x^3 + 24x + 13 \pmod{29}$ 上的一点，找到 $7P$ 。

```
>> multell ([1, 3], 7, 24, 13, 29)
ans =
    15   6
```

例 5：在前面例子的曲线上找出 $k(1, 3)$，$k = 1,2,3,\ldots,40$。

```
>> multsell ([1, 3], 40, 24, 13, 29)
ans =
    1     3
   11    10
   23    28
    0    10
   19     7
   18    19
   15     6
   20    24
    4    12
    4    17
   20     5
   15    23
   18    10
   19    22
    0    19
   23     1
   11    19
    1    26
   Inf  Inf
    1     3
   11    10
   23    28
    0    10
   19     7
   18    19
   15     6
   20    24
    4    12
    4    17
   20     5
```

```
15    23
18    10
19    22
 0    19
23     1
11    19
 1    26
Inf   Inf
 1     3
11    10
```

注意这些点每隔 19 个后是如何重复的。

例 6：前面 4 个例子都是工作在模素数 29 方式下的。如果模一个合数，由于在无穷大处可以对两个因子都取模或只对其中一个取模而不对另外一个取模，所以情况变得更复杂。因此，如果后一种情况出现我们就停止计算，并得出一个因子。例如，我们计算 $12P$，其中 $P=(1,3)$ 在椭圆曲线 $y^2\equiv x^3-5x+13(\mathrm{mod}\ 209)$ 上：

```
>> multell ( [1, 3] , 12,  -5, 13, 11*19)
Elliptic Curve addition produced a factor of n, factor =19
Multell found a factor of n and exited
ans  =
    []
```

现在继续计算连乘积，看能出现什么情况：

```
>> multsell ( [1, 3] , 12,  -5, 13, 11*19)
Elliptic Curve addition produced a factor of n, factor =19
Multsell ended early since it found a factor
ans =
         1    3
        91   27
       118  133
       148  182
        20   35
```

当我们计算 $6P$ 时，在无穷大模 19 处结束。让我们来看看模 209 的两个素因子（即 19 和 11）时发生了什么情况：

```
>> multsell ( [1, 3] , 20,  -5, 13, 19)
ans =
  1    3
 15    8
  4    0
 15   11
  1   16
Inf  Inf
  1    3
 15    8
  4    0
```

```
 15    11
  1    16
 Inf  Inf
  1     3
 15     8
  4     0
 15    11
  1    16
 Inf  Inf
  1     3
 15     8
> > multsell ( [1, 3] , 20, -5, 13, 11)
ans =
   1     3
   3     5
   8     1
   5     6
   9     2
   6    10
   2     0
   6     1
   9     9
   5     5
   8    10
   3     6
   1     8
  Inf  Inf
   1     3
   3     5
   8     1
   5     6
   9     2
   6    10
```

经历 6 步之后，到达了无限大模 19 的位置 ，但是用了 14 步才到达无限大模 11 的位置。为了找到6 P，我们反转一个数字，用0 对 19 取模并且非零数对 11 取模。这样做是不可能的，然而它产生了因子 19。这正是椭圆曲线因数分解法的基础。

例 7：使用椭圆曲线分解 193279。

解：首先，需要选择一些随机的椭圆曲线和每条曲线上的一个点。例如，取点 $P = (2,4)$ 和椭圆曲线

$$y^2 \equiv x^3 - 10x + b \pmod{193279}。$$

为使 P 位于曲线上，取 $b = 28$。同样我们还取

$$y^2 \equiv x^3 + 11x - 11, \quad P = (1,1),$$

$$y^2 \equiv x^3 + 17x - 14, \quad P = (1,2)。$$

现在计算点 P 的倍数。仿照 $p-1$ 的方法做，所以选择一个界限 B，令 $B = 12$，并计算 $B!P$。

```
>> multell ([2, 4], factorial (12), -10, 28, 193279)
Elliptic Curve addition produced a factor of n, factor =347
Multell found a factor of n and exited
ans =
     []
>> multell ([1, 1], factorial (12), 11, -11, 193279)
ans =
      13862   35249
>> multell ([1, 2], factorial (12), 17, -14, 193279)
Elliptic Curve addition produced a factor of n, factor =557
Multell found a factor of n and exited
ans =
     []
```

让我们更详细地分析在这些例子中发生了什么。

在第一个曲线上，$266P$ 结束在无穷大模 557 处，$35P$ 结束在无穷大模 347 处，因为 $266 = 2 \cdot 7 \cdot 19$，它是一个比 $B = 12$ 大的质因子，所以 $B!P$ 不是无穷大模 557 的，但是 35 可以除尽 $B!$，因此 $B!P$ 是无穷大模 347 的。

在第二个曲线上，$356P$ = 无穷大模 347 ，$561P$ = 无穷大模 557。因为 $356 = 4 \cdot 89$ 和 $561 = 3 \cdot 11 \cdot 17$ ，我们不能找到用这个曲线的因数分解。

第三个曲线是令人惊奇的。有 $331P$ = 无穷大模 347 和 $272P$ = 无穷大模 557。由于 331 是素数并且 $272 = 16 \cdot 17$，我们不可能用这个曲线找到因数分解。然而，巧的是在计算 B!P 时中间步骤仍产生了因数分解。下面就是所发生的。在第一步，程序要求增加点（184993，13462）和（20678，150484）。这两个点都是和 557 同余的，但与 347 不同余，因此，倾斜穿越这两点的线被定义为模 347 的，但却是 0/0 mod 557 的。当我们试着找出分母对 193279 取模的乘积逆元素时，最大公约数算法会产生因子 557，这一现象非常少有。

例 8：这里是如何产生 15.5 节中 ElGamal 加密体制的椭圆曲线的例子。欲知更多的细节，请参见原文。椭圆曲线是 $y^2 \equiv x^3 + 3x + 45 \pmod{8831}$，点是 $G = (4,11)$。艾丽斯的消息是点 $P_m = (5,1743)$。

鲍勃已经选择了秘密随机数 $a_B = 3$，而且已经计算了 a_bG：

```
>> multell ([4, 11], 3, 3, 45, 8831)
ans =
    413   1808
```

鲍勃公布了这个点。艾丽斯选择了随机数 $k = 8$，计算 kG 和 $P_m + k(a_BG)$：

```
>> multell ([4, 11], 8, 3, 45, 8831)
ans =
    5415   6321
>> addell ([5, 1743], multell ([413, 1808], 8, 3, 45, 8831), 3, 45, 8831)
```

```
ans =
    6626   3576
```

艾丽斯传送点（5415，6321）和（6626，3576）给鲍勃，她用 a_B 乘以第一个点：

```
>> multell ( [5415, 6321] , 3, 3, 45, 8831)
ans =
     673   146
```

然后鲍勃减去艾丽斯发送给他的最后一个点的结果。注意，他所做的减法是通过加第二个被丢弃点来实现的：

```
>> addell ( [6626, 3576],[673, -146] , 3, 45, 8831)
ans =
     5   1743
```

因此鲍勃正确地得到了艾丽斯的信息。

例 9：让我们用 15.5 节 DiffieHellman 密钥交换的例子中的方法再产生一些数据：椭圆曲线是 $y^2 \equiv x^3 + x + 7206 \pmod{7211}$，点是 $G = (3,5)$。艾丽斯选择她的秘密数 $N_A = 12$，鲍勃选择他的秘密数 $N_B = 23$。艾丽斯计算

```
>> multell ( [3, 5] , 12, 1, 7206, 7211)
ans =
   1794   6375
```

她发送点（1794，6375）给鲍勃。同时，鲍勃计算

```
>> multell ( [3, 5] , 23, 1, 7206, 7211)
ans =
    3861   1242
```

并发送点（3861，1242）给艾丽斯。艾丽斯用 N_A 乘以她所收到的值，鲍勃用 N_B 乘以他所收到的值：

```
>> multell ( [3861, 1242], 12, 1, 7206, 7211)
ans   =
     1472   2098
>> multell ( [1794, 6375], 23, 1, 7206, 7211)
ans =
    1472   2098
```

因此，艾丽丝和鲍勃得到了相同的密钥。

附录 D 进一步阅读的建议

追溯密码学的历史，最好的来源就是［Kahn］。

对于本书中需要进一步阅读的主题，以及相关的其他方面的主题内容，可参见［Stinson］，［Schneier］和［Menezes et al.］，这些书都有更广泛的参考价值。

在密码学中一个接近的重要的代数方法在［Koblitz］中给出。

更偏重协议实现和网络安全的一本书是［Stallings］。

当然 Internet 上也有包括关于密码学问题的信息资源。而且，学报 *CRYPTO*，*EUROCRYPT* 和 *ASIACRYPT*（出版于 Springer-Verlag 的 Computer Science 系列的 Lecture Notes）包含很多最新发展的内容。

参考文献

[Atkins et al.] D. Atkins, M. Graff, A. Lenstra, P. Leyland, "The magic words are squeamish ossifrage," *Advances in Cryptology — ASIACRYPT '94*, Lecture Notes in Computer Science 917, Springer-Verlag, 1995, pp. 263–277.

[Beker-Piper] H. Beker and F. Piper, *Cipher Systems: The Protection of Communications*, Wiley-Interscience, 1982.

[Berlekamp] E. R. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill, 1968.

[Blake et al.] I. Blake, G. Seroussi, N. Smart, *Elliptic Curves in Cryptography*, Cambridge University Press, 1999.

[Blom] R. Blom, "An optimal class of symmetric key generation schemes," *Advances in Cryptology — EUROCRYPT'84*, Lecture Notes in Computer Science 209, Springer-Verlag, 1985, pp. 335–338.

[Boneh] D. Boneh, "Twenty years of attacks on the RSA cryptosystem," *Amer. Math. Soc. Notices* 46 (1999), 203–213.

[Boneh et al.] D. Boneh, G. Durfee, and Y. Frankel, "An attack on RSA given a fraction of the private key bits," *Advances in Cryptology — ASIACRYPT '98*, Lecture Notes in Computer Science 1514, Springer-Verlag, 1998, pp. 25–34.

[Brands] S. Brands, "Untraceable off-line cash in wallets with observers," *Advances in Cryptology — CRYPTO'93*, Lecture Notes in Computer Science 773, Springer-Verlag, 1994, pp. 302–318.

[Campbell-Wiener] K. Campbell and M. Wiener, "DES is not a group," *Advances in Cryptology — CRYPTO '92*, Lecture Notes in Computer Science 740, Springer-Verlag, 1993, pp. 512–520.

[Chabaud] F. Chabaud, "On the security of some cryptosystems based on error-correcting codes," *Advances in Cryptology — EUROCRYPT'94*, Lecture Notes in Computer Science 950, Springer-Verlag, 1995, pp. 131–139.

[Cohen] H. Cohen, *A Course in Computational Number Theory*, Springer-Verlag, 1993.

[Coppersmith1] D. Coppersmith, "The Data Encryption Standard (DES) and its strength against attacks," *IBM Journal of Research and Development*, vol. 38, no. 3, May 1994, pp. 243–250.

[Coppersmith2] D. Coppersmith, "Small solutions to polynomial equations, and low exponent RSA vulnerabilities," *J. Cryptology* 10 (1997), 233–260.

[Cover-Thomas] T. Cover and J. Thomas, *Elements of Information Theory*, Wiley Series in Telecommunications, 1991.

[Crandall-Pomerance] R. Crandall and C. Pomerance, *Prime Numbers, a Computational Perspective*, Springer-Telos, 2000.

[Damgård et al.] I. Damgård, P. Landrock, and C. Pomerance, "Average case error estimates for the strong probable prime test," *Mathematics of Computation* 61 (1993), 177–194.

[Diffie-Hellman] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Transactions in Information Theory*, 22 (1976), 644-654.

[Ekert-Josza] A. Ekert and R. Jozsa, "Quantum computation and Shor's factoring algorithm," *Reviews of Modern Physics*, 68 (1996), 733–753.

[Fortune-Merritt] S. Fortune and M. Merritt, "Poker Protocols," *Advances in Cryptology — CRYPTO'84*, Lecture Notes in Computer Science 196, Springer-Verlag, 1985, pp. 454–464.

Bibliography

[Gaines] H. Gaines, *Cryptanalysis*, Dover Publications, 1956.

[Gallager] R. G. Gallager, *Information Theory and Reliable Communication*, Wiley, 1969.

[Gilmore] *Cracking DES: Secrets of Encryption Research, Wiretap Politics & Chip Design*, Electronic Frontier Foundation, J. Gilmore (editor), O'Reilly and Associates, 1998.

[Girault et al.] M. Girault, R. Cohen, and M. Campana, "A generalized birthday attack," *Advances in Cryptology — EUROCRYPT'88*, Lecture Notes in Computer Science 330, Springer-Verlag, 1988, pp. 129–156.

[Golomb] S. Golomb, *Shift Register Sequences*, 2nd ed., Aegean Park Press, 1982.

[Hughes et al.] R.J. Hughes et al., "Quantum Cryptography over Underground Optical Fibers," *Advances in Cryptology — CRYPTO'96*, Lecture Notes in Computer Science 1109, Springer-Verlag, 1996, pp. 329–342.

[Kahn] D. Kahn, *The Codebreakers*, 2nd ed., Scribner, 1996.

[Kilian-Rogaway] J. Kilian and P. Rogaway, "How to protect DES against exhaustive key search (an analysis of DESX)," *J. Cryptology* 14 (2001), 17–35.

[Koblitz] N. Koblitz, *Algebraic Aspects of Cryptography*, Springer-Verlag, 1998.

[Kocher] P. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," *Advances in Cryptology — CRYPTO '96*, Lecture Notes in Computer Science 1109, Springer, 1996, pp. 104–113.

[Kozaczuk] W. Kozaczuk, *Enigma: How the German Machine Cipher Was Broken, and How It Was Read by the Allies in World War Two*; edited and translated by Christopher Kasparek, Arms and Armour Press, London, 1984.

[Lin-Costello] S. Lin and D. J. Costello, Jr., *Error Control Coding: Fundamentals and Applications*, Prentice Hall, 1983.

[MacWilliams-Sloane] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, North-Holland, 1977.

[Matsui] M. Matsui, "Linear cryptanalysis method for DES cipher," *Advances in Cryptology — EUROCRYPT'93*, Lecture Notes in Computer Science 765, Springer-Verlag, 1994, pp. 386–397.

[Menezes et al.] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.

[Merkle-Hellman] R. Merkle and M. Hellman, "On the security of multiple encryption," *Communications of the ACM* 24 (1981), 465–467.

[Nelson-Gailly] M. Nelson and J.-L. Gailly, *The Data Compression Book*, M&T Books, 1996.

[Okamoto-Ohta] T. Okamoto and K. Ohta, "Universal electronic cash," *Advances in Cryptology — CRYPTO'91*, Lecture Notes in Computer Science 576, Springer-Verlag, 1992, pp. 324–337.

[Quisquater et al.] J.-J. Quisquater and L. Guillou, "How to explain zero-knowledge protocols to your children," *Advances in Cryptology — CRYPTO '89*, Lecture Notes in Computer Science 435, Springer-Verlag, 1990, pp. 628–631.

[Rieffel-Polak] E. Rieffel and W. Polak, "An Introduction to Quantum Computing for Non-Physicists," available at *xxx.lanl.gov/abs/quant-ph/9809016*.

[Schneier] B. Schneier, *Applied Cryptography*, 2nd ed., John Wiley, 1996.

[Shannon1] C. Shannon, "Communication theory of secrecy systems," *Bell Systems Technical Journal* 28 (1949), 656–715.

[Shannon2] C. Shannon, "A mathematical theory of communication," *Bell Systems Technical Journal*, 27 (1948), 379–423, 623–656.

Bibliography

[Stallings] W. Stallings, *Cryptography and Network Security: Principles and Practice*, 2nd ed., Prentice Hall, 1998.

[Stinson] D. Stinson, *Cryptography: Theory and Practice*, CRC Press, 1995.

[Thompson] T. Thompson, *From Error-Correcting Codes through Sphere Packings to Simple Groups*, Carus Mathematical Monographs, number 21, Mathematical Association of America, 1983.

[van der Lubbe] J. van der Lubbe, *Basic Methods of Cryptography*, Cambridge University Press, 1998.

[van Oorschot-Wiener] P. van Oorschot and M. Wiener, "A known-plaintext attack on two-key triple encryption," *Advances in Cryptology – EUROCRYPT '90*, Lecture Notes in Computer Science 473, Springer-Verlag, 1991, pp. 318–325.

[Welsh] D. Welsh, *Codes and Cryptography*, Oxford, 1988.

[Wicker] S. Wicker, *Error Control Systems for Digital Communication and Storage*, Prentice Hall, 1995.

[Wiener] M. Wiener, "Cryptanalysis of short RSA secret exponents," *IEEE Trans. Inform. Theory*, 36 (1990), 553–558.

[Williams] H. Williams, *Edouard Lucas and Primality Testing*, Wiley-Interscience, 1998.